Beating Brute Force for Compression Problems[*]

Shuichi Hirahara[†]
National Institute of Informatics
Tokyo, Japan
s_hirahara@nii.ac.jp

Rahul Ilango[‡]
Massachusetts Institute of Technology
Cambridge, USA
ilangorahul@gmail.com

R. Ryan Williams[§]
Massachusetts Institute of Technology
Cambridge, USA
rrw@mit.edu

ABSTRACT

A *compression problem* is defined with respect to an efficient encoding function f; given a string x, our task is to find the shortest y such that $f(y) = x$. The obvious brute-force algorithm for solving this compression task on n-bit strings runs in time $O(2^\ell \cdot t(n))$, where ℓ is the length of the shortest description y and $t(n)$ is the time complexity of f when it prints n-bit output.

We prove that every compression problem has a Boolean circuit family which finds short descriptions more efficiently than brute force. In particular, our circuits have size $2^{4\ell/5} \cdot \mathsf{poly}(t(n))$, which is significantly more efficient for all $\ell \gg \log(t(n))$. Our construction builds on Fiat-Naor's data structure for function inversion [SICOMP 1999]: we show how to carefully modify their data structure so that it can be nontrivially implemented using Boolean circuits, and we show how to utilize hashing so that the circuit size is only exponential in the description length.

As a consequence, the Minimum Circuit Size Problem for generic fan-in two circuits of size $s(n)$ on truth tables of size 2^n can be solved by circuits of size $2^{\frac{4}{5}\cdot w+o(w)} \cdot \mathsf{poly}(2^n)$, where $w = s(n)\log_2(s(n)+n)$. This improves over the brute-force approach of trying all possible size-$s(n)$ circuits for all $s(n) \geq n$. Similarly, the task of computing a short description of a string x when its K^t-complexity is at most ℓ, has circuits of size $2^{\frac{4}{5}\ell} \cdot \mathsf{poly}(t)$. We also give nontrivial circuits for computing Kt complexity on average, and for solving NP relations with "compressible" instance-witness pairs.

CCS CONCEPTS

• **Theory of computation** → **Circuit complexity**; *Cryptographic primitives.*

KEYWORDS

compression, circuit complexity, function inversion

ACM Reference Format:
Shuichi Hirahara, Rahul Ilango, and R. Ryan Williams. 2024. Beating Brute Force for Compression Problems. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649778

[*]The full version of this paper is available at https://eccc.weizmann.ac.il/report/2023/171/.
[†]Supported by JST, PRESTO Grant Number JPMJPR2024, Japan.
[‡]Supported by an NSF Graduate Fellowship.
[§]Supported by NSF CCF-2127597.

1 INTRODUCTION

Are there NP problems that require brute-force search in order to be solved? This basic question is one of the prime motivations behind P versus NP. It is a longstanding open question to determine if (for example) there are any *non-trivial* algorithms for the CircuitSAT problem which run in $2^n/n^{\omega(1)}$ time where n is the number of inputs. Such an algorithm, besides being interesting in its own right, would also settle important open questions in complexity theory: for example, it would imply that NEXP does not have polynomial-size circuits [50]. As of now, there has been no progress on beating brute force for CircuitSAT on circuits of size greater than $5n$ (but for small enough circuits, some algorithms are known, e.g. [12, 22]).

In this paper, we show how to generically improve over brute force using *non-uniformity*. In recent years, NP problems based on *compression* have been extensively studied, and are at the core of a new topic in TCS called "meta-complexity." Let us formally define what we mean by a compression problem.

Definition 1.1 (Compression Problem). *Let* Eval : $\{0,1\}^\star \to \{0,1\}^\star$ *be a polynomial-time computable. The* Eval *compression problem is:*

- ***Input:*** *a string* $x \in \{0,1\}^n$ *and a size parameter* s
- ***Output:*** *return a string* y *with* $|y| \leq s$ *such that* $\mathrm{Eval}(y) = x$, *or output* $\perp$ *if no such* y *exists.*

Two prominent examples of compression problems are the Minimum Circuit Size Problem (MCSP) and time-bounded Kolmogorov complexity (MINKT). For MCSP, Eval takes the description of a circuit and outputs its truth table, where the length of the description directly correlates with the number of gates in the circuit. For MINKT, we are given a string x, a time bound t, and an integer k, and we wish to know if there is a program of length at most k that outputs x in at most t steps. For any fixed time function t, the Eval function takes a program p as input and runs p for t steps, and we wish to minimize the length of the program. These two compression problems have many interesting connections to complexity theory [44], circuit complexity [9, 41], average-case complexity [24, 26], cryptography [31, 32, 36, 37, 45], and learning theory [7, 28].

1.1 Our Results

The main result of this paper is the construction of a (non-uniform) circuit family which can solve all generic compression problems significantly faster than the obvious brute-force enumeration of all possible programs up to a given length.

Theorem 1.2. *Let* Eval *denote a compression problem. There is a circuit family $\{C_{n,s}\}$ such that for all $n, s \in \mathbb{N}$, $C_{n,s}$ solves the compression problem for* Eval *on all strings of length n with descriptions of length at most s, and the size of $C_{n,s}$ is $2^{\frac{4}{5}\cdot s} \cdot \mathrm{poly}(n,s)$. Furthermore, $C_{n,s}(x)$ prints a description of length at most s for the input x of length n, whenever such a description exists.*

Theorem 1.2 has interesting consequences for meta-complexity problems.

Smaller Circuits for MCSP. The first major consequence of Theorem 1.2 is a Boolean circuit family for MCSP that decisively beats exhaustive search. It has been conjectured since the 1950s that any algorithm solving MCSP must exhaustively search over all possible circuits (see Trakhtenbrot [46] for a fascinating history of these "perebor" conjectures). Our results refute this conjecture when we are allowed non-uniform circuits as our algorithmic model.

In more detail, we consider circuits of fan-in two over any desired basis. Let Search-MCSP$[s(n)]$ be the search problem:

> Given a truth table T of length 2^n, determine if the function $f : \{0,1\}^n \to \{0,1\}$ represented by T has a circuit of at most $s(n)$ gates, and if so, produce an encoding of such a circuit.

The obvious algorithm for this search problem requires time $2^{O(s(n)\log_2 s(n))} \cdot 2^n \cdot \mathrm{poly}(s(n))$: enumerate over all $2^{O(s(n)\log_2 s(n))}$ circuits C of size $s(n)$ with n inputs, and evaluate C on all 2^n possible inputs in $2^n \cdot \mathrm{poly}(s(n))$ time. The MCSP problem and its search version are believed to be NP-hard, but their complexities remain (infamously) open (cf. [27, 30] for some recent developments).

Using tight and efficient circuit encodings, we obtain an improvement over the trivial enumeration algorithm for all circuit sizes $s(n) \geq n$.

Theorem 1.3. *For all size functions $s(n)$,* Search-MCSP$[s(n)]$ *on truth tables of size 2^n can be solved by circuits of size $2^{\frac{4}{5}\cdot w + o(w)} \cdot \mathrm{poly}(2^n)$, where $w = s(n)\log_2(s(n)+n)$.*

Thus for example, there is a Boolean circuit C of size only

$$2^{\frac{4}{5}n^2\log(n) + o(n^2\log(n))} \cdot \mathrm{poly}(2^n)$$

which, given any truth table of length 2^n as input, C outputs the description of a circuit with n^2 gates for the truth table, whenever such a circuit exists.

Smaller Circuits for MINKT. Theorem 1.2 is very general, and applies in a wide range of compression settings. As another example, we can apply Theorem 1.2 to show that for every fixed time function $t(n)$, there are nontrivial circuits computing the $\mathrm{K}^{t(n)}$-complexity of n-bit strings.

Theorem 1.4. *For every time function $t : \mathbb{N} \to \mathbb{N}$ with $t(n) \geq n$, and parameters $e, n \in \mathbb{N}$, there is a circuit family that given any n-bit input x, outputs a program y of length at most e such that y prints the string x in at most $t(n)$ steps if such a y exists. The circuit family has size $2^{\frac{4}{5}e} \cdot \mathrm{poly}(t(n))$.*

This answers an open question of Ren and Santhanam [45], who, based on connections between K^t and one-way functions, suggested that there may be a non-trivial circuit for solving K^t. Independently of this work, the same result (for the case $e = n$) was obtained by Mazor and Pass [38].

Smaller Circuits for Compressible Instances of NP *Relations.* Our main result can also be applied to construct non-trivial circuits for *compressible* instances of NP relations. Formally, let $R \subseteq \{0,1\}^\star \times \{0,1\}^\star$ be any polynomial-time computable relation. We wish to solve the following task, where $n, p \in \mathbb{N}$ are parameters.

> **Compressible-R:** Given a string x of length n, if there is a program of size p which prints the pair (x, y) in $\mathrm{poly}(n)$ time such that $(x,y) \in R$, find a y' such that $(x, y') \in R$.

The obvious brute-force algorithm for **Compressible-R** runs in $2^p \cdot \mathrm{poly}(n)$ time, by enumerating over all programs of length p, running each program in $\mathrm{poly}(n)$ time, and testing their output in $\mathrm{poly}(n)$ time.

Theorem 1.5. **Compressible-R** *can be solved by circuits of size $2^{\frac{4}{5}\cdot p} \cdot \mathrm{poly}(n)$.*

Smaller Circuits for MKtP *on Average.* Levin's Kt-complexity of a string x is defined as the minimum, over all $t \in \mathbb{N}$ and a program d, of $|d| + \log t$ such that d prints x in time t. The Minimum Kt Complexity Problem (MKtP) asks to compute $\mathrm{Kt}(x)$ on input x. By exhaustive search, MKtP can be solved in time $2^n\mathrm{poly}(n)$. We show that there exists a non-trivial circuit that computes $\mathrm{Kt}(x)$ on *most* instances drawn from any efficiently computable distribution. A distribution $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$ is said to be *$t(n)$-time-computable* [6, 34] if there exists a $t(n)$-time algorithm that, on input $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, computes the cumulative function of $\mathcal{D}_n$ on x. (That is, given a string x, we can compute the probability that a random string from $\mathcal{D}_n$ is at most x, under the natural ordering on n-bit strings, in $t(n)$ time.)

Theorem 1.6. *For all functions $s(n)$ and $t(n) \geq n$, there exists a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ of size $2^{\frac{4}{5}s(n)} \cdot \mathrm{poly}(t(n))$ such that for any $t(n)$-time-computable distribution $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$ over $\{0,1\}^n$, for all large $n \in \mathbb{N}$, with probability at least $1 - \frac{1}{t(n)}$ over a random input x drawn from $\mathcal{D}_n$, on input x, the circuit C_n outputs a program y and $t \in \mathbb{N}$ such that y prints the string x in t steps and $|y| + \log t \leq s(n)$ if $\mathrm{Kt}(x) \leq s(n)$.*

1.2 Intuition

The starting point of our approach comes from cryptography, namely the problem of inverting a function $f : \{0,1\}^n \to \{0,1\}^n$ using a minimal number of black-box calls to f. In particular, given a $y \in \{0,1\}^n$, we wish to find an x such that $f(x) = y$. Following the pioneering work of [23] on data structures for inverting random functions, Fiat and Naor [17] presented data structures with a rigorous time-space tradeoff. In particular, for $f : \{0,1\}^n \to \{0,1\}^n$ construed as an oracle, Fiat and Naor show that one can construct a data structure with S bits of memory that can be queried for function inversion in time T, where $T \cdot S^3 = 2^{3n} \cdot \mathrm{poly}(n)$. Setting $T = S = 2^{3n/4}$, for every fixed f we obtain a data structure storing $2^{3n/4} \cdot \mathrm{poly}(n)$ bits such that, given any y in the range of f, the data structure will output a pre-image of y in about $2^{3n/4}$ steps.

Given the power and generality of function inversion, one might wonder if it can be used to build a non-trivial "data structure" for solving an NP-complete problem like CircuitSAT faster than 2^n time. A natural first attempt is to set f to take a circuit C and a string w as input, outputting C if $C(w) = 1$ and $\perp$ otherwise. Given a worst-case inverter for f, one could attempt to solve CircuitSAT by attempting to invert f on C.

One major difficulty in carrying out such an approach is that the input length to the function f is too large. If the circuit C has n inputs, the witness w and C both need at least n bits to describe. (Note that, if the circuits or the witnesses could be described in say $0.9n$ bits, then we could *already* trivially improve over 2^n time by simply storing a $2^{0.9n}$-size lookup table over the circuits, or the witnesses.) Thus, f takes at least $2n$ input bits, and so the inversion circuit obtained from Fiat-Naor has size at least $2^{6n/4} \gg 2^n$, which is worse than brute force.

In contrast, the witness in a compression problem *already encodes the input*, so the previous paragraph is not an issue. Indeed, if we simply apply Fiat-Naor's function inversion to the Eval function of a compression problem, we immediately obtain a data structure that takes $2^{3n/4}$ space and can compress n-bit queries in $2^{3n/4}$ time for an arbitrary compression problem.

However, there remain two issues in the generality of this simple approach, which we overcome.

(1) The first is a technical algorithmic issue. The data structure of Fiat-Naor requires *random access* to its storage, and the most obvious way of converting such a data structure to a circuit would blow up the size by an intolerable amount (cf. footnote 2 of [15]). This issue is what prevented Ren and Santhanam [45] from giving non-trivial circuits for MINKT via their equivalence between one-way functions (with exponential security) and the hardness of MINKT. (Note: this tight connection is not known for other compression problems, like MCSP.) We show how to implement Fiat-Naor with standard Boolean circuits, at the cost of a slightly larger circuit (the exponent in the circuit size becomes $4/5$, rather than the exponent of $3/4$). This requires us to be very careful about certain parts of the inversion procedure; we have to worry over details that Fiat and Naor did not worry about. As a result, we have to adapt both the inversion procedure and the analysis of it, in order to achieve our circuit size bound. Roughly speaking, by performing the necessary table lookups in large enough query batches, and adjusting the sizes of lookup tables in the analysis, it is possible to design nontrivial circuits that simulate the data structure by performing variations on sorting.
The independent work of Mazor and Pass [38] also used sorting ideas to implement Fiat-Naor with Boolean circuits.

(2) The second problem is that Fiat-Naor only provides a $2^{3n/4}$ time data structure and algorithm for inversion, *where n is the length of the input we wish to invert.* Such a bound is useless when our desired description length ℓ is less than $3n/4$, in which case the 2^ℓ cost of brute-force enumeration is faster than Fiat-Naor! (For example, in the case of MCSP, we would obtain a $2^{0.75 \cdot 2^n}$-time data structure for solving MCSP on truth tables of length 2^n.) That is, naively applying Fiat-Naor only yields an improvement over exhaustive search in the case where the complexity of the string is already very close to the maximum possible.
In order to beat the 2^ℓ exhaustive search over descriptions of length ℓ, for *every* ℓ, we have to take a different approach. Rather than inverting a function that maps n bits into n bits, we need to invert an Eval function which maps e bits (the compressed length) into n bits (the decompressed length). Furthermore, we want the cost of inverting our function to be exponential *only in* e, and polynomial in n (and the circuit size of Eval).
We achieve this by (pairwise independent) hashing: we consider a new function which applies Eval to an encoding of length $e + O(1)$, and hashes its n-bit result to a string of length $e + O(1)$. Starting from this idea, we show that the problem of inverting a function f from e bits to n bits can be *generically reduced* to the problem of inverting a function f' from $e + 1$ bits to $e + 1$ bits; then, we can apply our circuits for function inversion to the function f'.

Our hashing reduction was inspired by the literature on *hardness magnification* [9, 10, 13, 25, 39–41], a phenomenon in which a (seemingly) weak circuit lower bound for a specific problem is shown to imply a breakthrough result in complexity theory, such as P $\neq$ NP. For example, McKay et al. [39] showed that if there is *any* $c \geq 1$ such that Search-MCSP$[n^c]$ does not have $\widetilde{O}(N)$-size circuits for $N = 2^n$, then NP $\not\subseteq$ P/poly. Such a result is proved by the contrapositive: Assuming NP $\subseteq$ P/poly, one builds a $\widetilde{O}(N)$-size circuit for Search-MCSP$[n^c]$, using the property that Search-MCSP$[n^c]$ is reducible to instances of a PH problem of size $\widetilde{O}(n^c) \ll N$. The primary difference between hardness magnification and our results is that we use the *unconditional* construction of the non-uniform algorithm for function inversion, instead of *hypothetical* upper bounds, such as NP $\subseteq$ P/poly. We remark that a similar hashing trick was used in the context of function inversion by Corrigan-Gibbs and Kogan [14].

The theory of function inversion has recently seen a renewed interest; works on function inversion in theoretical cryptography improve the known time-space tradeoffs in different computational models and settings [5, 8, 15, 16, 21], and find other interesting consequences of function inversion [14, 20]. In particular, [20] show how to use function inversion to refute a data structure conjecture on 3SUM in fine-grained complexity.

2 PRELIMINARIES

For a function $f : \{0, 1\}^n \to \{0, 1\}^n$ and a non-negative integer p, we let f^p denote the composition of f with itself p times. Our convention is that f^0 is the identity function.

For our circuits solving Search-MCSP, we will utilize the fact that there are very efficient encodings of fan-in two circuits over any basis.

LEMMA 2.1 (EFFICIENT ENCODING OF CIRCUITS [19]). *There is a polynomial time algorithm* Enc *such that the following holds. For every circuit C of size s on n-inputs, there is a string x of length $(1 + o(1))s \log_2(s + n)$ such that* $\mathrm{Enc}(x)$ *outputs a description of a circuit of size s computing the same function as C.*[1]

[1]Although we do not need this property for our purposes, such an encoding x can be computed from any given C in polynomial-time.

One of the primary bottlenecks in implementing Fiat-Naor's function inversion in the Boolean circuit model is that, in order to beat exhaustive search, their procedure apparently requires unit-time random access to a lookup table (see footnote 2 of [15] for a discussion on this point). By adjusting various parameters in their data structure, we can simulate their lookup tables in an efficient way by only querying tables on batches of queries. The formal theorem we need is the following.

Theorem 2.2 (Batch Queries to a Lookup Table). *Let $m, n \geq 1$. There is a circuit C with $2mn$ inputs and m outputs of size $O(m \cdot n \cdot \log^2(mn))$, such that given a list of m strings $X_1, \ldots, X_m \in \{0,1\}^n$, and a list of m queries $X'_1, \ldots, X'_m \in \{0,1\}^n$, C returns bits $b_1, \ldots, b_m$ where $b_i = 1$ if and only if $X'_i \in \{X_1, \ldots, X_m\}$.*

Proof. Our circuit generalizes a result of W. J. Paul ([42], Lemma 2) on efficiently evaluating hard Boolean functions on multiple inputs. First we describe a multitape Turing machine for computing the task which runs in $O(m \cdot n \cdot \log m)$ time. Then, it follows from Pippenger-Fischer [43] that there is a circuit family of size at most $O(m \cdot n \cdot \log^2(mn))$ for the task.

Suppose for simplicity that our Turing machine M is given the n-bit strings $X_1, \ldots, X_m$ on one tape, and the queries $X'_1, \ldots, X'_m$ on another tape. (This can be achieved with a linear overhead.) Our machine M removes any duplicates from the list $X_1, \ldots, X_m$, by sorting the list, sweeping across the sorted order, and comparing adjacent strings, copying the *distinct* strings over to another tape. This takes $O(mn \log m)$ time: $O(m \log m)$ comparisons where each comparison costs $O(n)$ time.

Our machine M then maps each distinct X_i to the string $X_i \circ 0$: X_i concatenated with a zero. Similarly, each X'_i is mapped to $X'_i \circ 1 \circ i$. Next, M sorts the list of $2m$ items

$$\{X_i \circ 0, X'_i \circ 1 \circ i \mid i \in [m]\}.$$

This takes $O(mn \log m)$ time, as in the previous paragraph.

Next, M sweeps across the sorted order of $2m$ elements, from left to right, processing the strings. In particular, the sorted order consists of contiguous blocks of two possible kinds:

(1) $X \circ 0, X \circ 1 \circ i_1, \ldots, X \circ 1 \circ i_\ell$, for some $\ell \in \{0, 1, \ldots, n\}$, where each X is unique (as we have removed duplicates).
(2) $X \circ 1 \circ i_1, \ldots, X \circ 1 \circ i_\ell$, for some $\ell \in \{1, \ldots, n\}$ (with no prefix of the form $X \circ 0$).

We can ignore blocks of the form (2); since there is no $X \circ 0$ prefix to the block, there is no matching string in $X_1, \ldots, X_m$. For all blocks of the form (1), we write the $O(\log m)$-bit indices $i_1, \ldots, i_\ell$ on a separate tape: note that all i_j are indices in the m-bit output vector that must be 1. This step takes at most $O(mn \log m)$ time in total: for each block of ℓ items in the sorted order, it takes $O(\ell \cdot n)$ time to form the block by comparing strings (find where the block ends), and it takes $O(\ell n + \ell \log m)$ time to sweep through the block and write down the corresponding $O(\log m)$-bit index for each item in the block. (The sum of all block lengths is at most $2m$.)

Finally, the machine M processes the list of $O(\log m)$-bit indices to get the m-bit output. In particular, M sorts the $O(m)$-length list, removing duplicates. On a separate tape, M sweeps along m cells to print 0 or 1 for each bit i of the output, based on whether i is in the sorted list of indices. This final step takes $O(m \log m)$ time. □

We will also need a slight generalization of batch lookup, in which secondary information in the list can also be returned:

Theorem 2.3 (Batch Queries to Lookup Tables With Side Information). *Let $m, n, K \geq 1$. There is a circuit C with $O(mn)$ inputs and m outputs of size $O(m \cdot n \cdot \log^2(mn) + K \cdot n)$, such that given a list of m pairs $(X_1, Y_1) \ldots, (X_m, Y_m) \in \{0,1\}^n \times \{0,1\}^n$, a list of m queries $(p_1, X'_1), \ldots, (p_m, X'_m) \in [2^n] \times \{0,1\}^n$, and an integer upper bound $K \in [2^n]$, C returns a set of $t \leq K$ triples $\{(p_j, X'_{p_j}, Y_{p_j})\}$ such that $X'_{p_j} \in \{X_1, \ldots, X_m\}$ for all j and p_j is* minimal*, whenever such t triples exist.*

Proof (Sketch). We modify the Turing machine in the previous proof to sort the pairs $(X_1, Y_1) \ldots, (X_m, Y_m)$ according to the keys $X_1, \ldots, X_m$, and to sort the list $(p_1, X'_1), \ldots, (p_m, X'_m)$ according to the *primary* keys $X'_1, \ldots, X'_m$ and *secondary* keys $p_1, \ldots, p_m$. (First we sort according to the X'_i's, then we break ties by sorting according to the p_j's.) We merge the two sorted lists, as in the previous proof. When M sweeps across the merged sorted order, if a query pair (p_j, X'_j) matches a list pair (X'_j, Y_j), then M prints the entire triple (p_j, X'_j, Y_j) to an extra tape, ignoring later triples of the form (p', X'_j, Y_j) in the same block (recall that M is required to only print triples such that p_j is minimal). The machine M halts whenever K triples have been output, or the entire sorted list has been processed, whichever comes first. □

Some of the functions in Fiat-Naor's function inversion (namely, the k-wise independent hash functions) can be described by univariate polynomials. Fiat-Naor speeds up their evaluation in an amortized sense, by appealing to FFT. We will use the fact that FFT can also be efficiently simulated in the arithmetic circuit model:

Theorem 2.4 (Multipoint Evaluation of Polynomials, [18], see also [49]). *Let $\mathbb{F}$ be a field of characteristic two, and let $P \in \mathbb{F}[x]$ have degree d. There is an $\mathbb{F}_2$-arithmetic circuit C_P with d inputs and d outputs of size $d \cdot \mathrm{poly}(\log d)$ that, given $x_1, \ldots, x_d \in \mathbb{F}$, outputs $P(x_1), \ldots, P(x_d)$.*

3 MORE EFFICIENT CIRCUITS FOR COMPRESSION PROBLEMS

We now turn to our constructions of smaller circuits for compression problems. To start, we show how circuits of size about c^n for inverting functions from $\{0,1\}^n$ to $\{0,1\}^n$ can be used to obtain circuits of size about $c^e \cdot \mathrm{poly}(n)$ for inverting functions from $\{0,1\}^e$ to $\{0,1\}^n$, where $e \ll n$. (The latter case is the more relevant setting for compression problems, where e is the length of a short description and n is the length of the input.)

3.1 Obtaining Fixed-Parameter Tractable Circuits for General Compression

Suppose we are given a circuit for a function $\mathrm{Eval} : \{0,1\}^e \to \{0,1\}^n$ where $n \gg e$, so that Eval can be viewed as a "decompression" procedure mapping e-bit strings into longer n-bit strings. Given x of length n, the obvious brute-force strategy for finding a description y of length e such that $\mathrm{Eval}(y) = x$ requires about $2^e \cdot s$ time, where s is the evaluation time for Eval. Our goal in Theorem 1.2 is to construct $2^{4e/5} \cdot \mathrm{poly}(s)$-size circuits, strictly improving on the obvious bound in the exponent.

First, we observe that one cannot directly achieve such a bound by using circuits for Fiat-Naor function inversion. Fiat-Naor is designed to invert functions from (say) $\{0,1\}^n$ to $\{0,1\}^n$, where the domain and co-domain must be the same size, and the circuit size bound we can hope to achieve will have the form $2^{\delta n} \cdot \mathrm{poly}(n)$ for some $\delta \in (0,1)$. When $e \ll n$ (the interesting case of Eval!), this size bound is already much worse than the obvious brute-force cost of 2^e. We need to choose a different function to invert, one that maps $e + O(1)$ bits to $e + O(1)$ bits, in order to achieve a size bound of the form $2^{\delta e} \cdot \mathrm{poly}(n)$ for some $\delta \in (0,1)$.

Given the function Eval mapping e bits to n bits, we will use pairwise-independent hashing to show how to reduce the inversion problem for Eval to the inversion problem for a related function $f : \{0,1\}^{e+1} \to \{0,1\}^{e+1}$, so that given circuits of size $S(n)$ for function inversion, we can produce circuits of size $S(e+1) \cdot \mathrm{poly}(s)$ for inverting Eval.

Theorem 3.1. *Let $s(n) \geq n$. Suppose that for functions $g : \{0,1\}^n \to \{0,1\}^n$ with size-s circuits, there are circuits for inverting g which have size $S(n) \cdot \mathrm{poly}(s)$. Then for every $e \leq n$, there are circuits for inverting any function* Eval : $\{0,1\}^e \to \{0,1\}^n$ *of size $S(e+1) \cdot \mathrm{poly}(s)$, where $s(n) \geq n$ is the circuit size of* Eval.

Proof. Let $\mathcal{H} = \{h_i : \{0,1\}^n \to \{0,1\}^{e+1}\}$ be a family of pairwise-independent hash functions. In the following, we will just require the standard fact that there are hash families $\mathcal{H}$ such that every $h_i \in \mathcal{H}$ has a $\mathrm{poly}(n)$-size circuit (see for example [4], p.152–153).

Claim 3.2. *Suppose $x \in \{0,1\}^n$ has description length e under* Eval. *Drawing a uniform random $h \in \mathcal{H}$, the probability that x is the* unique *string with description length e in the preimage $h^{-1}(h(x))$ is at least $1/2$.*

Proof. Let S be the set of all n-bit strings x such that $\mathrm{Eval}(y) = x$ for some y which is e bits long. Fix a string $x \in S$, and note that $|S| \leq 2^e$. We want to lower bound the probability that a randomly chosen $h \sim \mathcal{H}$ "isolates" x from all other strings in S. The analysis is similar to proofs of the Valiant-Vazirani Lemma [4, 48], but with a different union bound: instead of fixing a target hash value (e.g., 0^{e+1}) and union-bounding over all possible $x \in S$, we fix the $x \in S$ and union-bound over possible $a \in \{0,1\}^{e+1}$.

Fix a particular $a \in \{0,1\}^{e+1}$. We have:

$$\begin{aligned}&\Pr_{h\sim\mathcal{H}}[h(x) = a \wedge (\forall y \in S - \{x\}) h(y) \neq a]\\ =&\frac{1}{2^{e+1}} \cdot \Pr[(\forall y \in S - \{x\}) h(y) \neq a \mid h(x) = a]\\ =&\frac{1}{2^{e+1}} \cdot (1 - \Pr[(\exists y \in S - \{x\}) h(y) = a \mid h(x) = a])\\ \geq&\frac{1}{2^{e+1}} \cdot \left(1 - \frac{2^e - 1}{2^{e+1}}\right) > \frac{1}{2^{e+2}}.\end{aligned}$$

Now, for each choice of $a \in \{0,1\}^{e+1}$, the 2^{e+1} events

$$[h(x) = a \wedge (\forall y \in S - \{x\}) h(y) \neq a]$$

are all disjoint. Therefore the probability there is *some* string $a \in \{0,1\}^{e+1}$ such that $h(x) = a$ and all other strings in $S - \{x\}$ do not hash to a is at least $1/2$. □

Using the claim, we can search for strings of description length e using function inversion. Let $\mathrm{Eval}' : \{0,1\}^{e+1} \to \{0,1\}^n$ be the procedure that ignores its last bit and evaluates $\mathrm{Eval} : \{0,1\}^e \to \{0,1\}^n$ on the remainder.

Compression From Function Inversion

```
Draw a random $h \in \mathcal{H}$.
Define the function $f : \{0,1\}^{e+1} \to \{0,1\}^{e+1}$ by $f(z) := h(\mathrm{Eval}'(z))$.
Given an input $x$ to compress:
    Try to invert $f$ on the $(e+1)$-bit string $h(x)$.
    If inversion finds $z = yb \in \{0,1\}^{e+1}$ with $|y| = e$, $|b| = 1$ such that $\mathrm{Eval}(y) = x$, return $y$.
    Return **Fail**.
```

Clearly, if x does not have a description of length e, then the above procedure always fails. Suppose x has a description of length e. For any $z = yb$ such that $f(z) = h(x)$, we have

$$h(x) = f(z) = h(\mathrm{Eval}'(z)) = h(\mathrm{Eval}(y)).$$

By the claim, with probability at least $1/2$, x is the *only* string with a description of length e in the preimage of $h^{-1}(h(x))$. Therefore with probability at least $1/2$, there is a preimage $z = yb$ of $f(x)$ and y is a length-e description of x.

We now analyze the efficiency of the procedure. Assume that for functions $g : \{0,1\}^n \to \{0,1\}^n$ with size-s circuits, there are circuits for inverting g which have size $S(n) \cdot \mathrm{poly}(s)$. Then the above procedure can be implemented with circuits of size $S(e+1) \cdot \mathrm{poly}(s)$, where $s \geq n$ upper bounds the circuit size of Eval. (Recall every $h \in \mathcal{H}$ has a polynomial-size circuit.)

The above describes a distribution of circuits for inverting Eval (based on the choice of the hash function h). A deterministic circuit can be constructed in a standard way, by simply taking $O(e) \leq O(n)$ random circuits from the distribution, and applying the union bound over all $O(2^e)$ strings of description length at most e. This introduces another multiplicative factor of at most $O(n)$ to the size. □

3.2 Warm-Up: Efficient Circuits for Inverting Cyclic Permutations

Next, we turn to constructing more efficient circuits for inverting functions from $\{0,1\}^n$ to $\{0,1\}^n$. As a warm-up, we start with circuits for inverting cyclic permutations, following the major insight of Hellman [23]. (Such circuits can be easily generalized to all permutations, using the fact that every permutation is a union of disjoint cycles.) Let $\pi : \{0,1\}^n \to \{0,1\}^n$ be a permutation. For $i \in \{0, \ldots, 2^n - 1\}$, let $y_i = \pi^i(0^n)$. π being cyclic means the list $y_0, \ldots y_{2^n - 1}$ contains no duplicates.

Then Figure 1 is a simple procedure for inverting π. Let k be a parameter we set later.

The correctness of this algorithm follows from the fact that $\pi^{k-p-1}(y_{j-k}) = \pi^{-p-1}(y_j) = \pi^{-1}(y)$.

The algorithm uses space about $\frac{2^n}{k}$ and takes time about k (assuming, for simplicity, we can compute π for free), so setting $k = 2^{n/2}$

```
Inverting Cyclic Permutations

Preprocessing: In a lookup table, store $(j, y_j)$ for all multiples $j$ of $k$.

Procedure: Given $y \in \{0,1\}^n$ to invert,
  • For $p \in \{0, \dots, k-1\}$:
    (1) Using the lookup table, check if $\pi^p(y) = y_j$ for some $j$ that is a multiple of $k$
    (2) If so, output $\pi^{k-p-1}(y_{j-k})$
```

Figure 1: Hellman's algorithm

yields an inversion procedure running in time and space $2^{n/2}$ on a (non-uniform) random access Turing Machine.

Can we implement this as a circuit? The main difference between the circuit model and the random access model is that accessing a bit from S-bits of storage in a circuit requires size roughly S (compared to cost $O(\log S)$ in the Turing Machine setting). Thus, the naive bound on the size of a circuit inverting cyclic permutations is roughly $k \cdot \frac{2^n}{k} \approx 2^n$ which gives no savings.

Luckily, this issue can be fixed by "batching" queries to memory. In particular, using Theorem 2.2, one can answer $2^{n/2}$ (non-adaptive) queries to a lookup table of size $2^{n/2}$ with a circuit of size roughly $2^{n/2}$.

Now observe the lookup queries in the inversion algorithm for cyclic permutations can be made non-adaptive: first calculate $\pi^p(y)$ for all $p \in \{0, \dots k-1\}$ and then query the lookup table on all of them at once. Using these ideas, one can indeed invert π with a circuit of size about $2^{n/2}$.

3.3 Implementing Fiat-Naor with Efficient Circuits

We now consider the general case of arbitrary function inversion for functions with small circuits, with the goal of proving Theorem 1.2. We begin by recalling the inversion algorithm of Fiat-Naor. The algorithm is parameterized by the following values.

Parameters.

- ℓ (the number of functions g_i we use)
- m (the number of checkpoints, i.e., the cardinality of the lookup table T_i for all i)
- t (the length of our walks)
- $|A|$ (the length of our lookup table of high degree points)
- k (the k-wise independence of our hash functions)
- Notation: (shorthand for values induced by a choice of parameters)
  - $N' = 2^n - \max_{S \subseteq \{0,1\}^n : |S| = |A|} |f^{-1}(S)|$ (the "effective" domain after choosing A)
  - $J = (n + \log t) \frac{2^n}{N'}$ (the number of times we can "resample" from a g_i function)

The algorithm has a randomized preprocessing step, where one builds several lookup tables. This is the only part of the algorithm that is randomized (in particular, the randomness is used to select k-wise independent functions).

```
(Randomized) Preprocessing for Inverting $f : \{0,1\}^n \to \{0,1\}^n$

  (1) Create the lookup table
        $A = \{(x, f(x)) : x \text{ is the lex. first preimage of } f(x) \in S\}$
      where $S$ is the set of size[a] $|A|$ that maximizes[b] $|f^{-1}(S)|$.
      We say $y$ is in $A$ if $(x, y) \in A$ for some $x$.
  (2) Sample $\ell$ many $k$-wise independent functions $g_1, \dots, g_\ell : \{0,1\}^n \times [J] \to \{0,1\}^n$.
  (3) Notation: For each $i \in [\ell]$,
      • $g_i^\star(x) = \begin{cases} g_i(x,j), & \text{for least } j \text{ with } f(g_i(x,j)) \notin A \\ \bot, & \text{if no such } j \text{ exists.} \end{cases}$
      • $h_i(x) = g_i^\star(f(x))$ (define $f(\bot) = \bot$ and $g_i^\star(\bot) = \bot$).
  (4) For each $i \in [\ell]$ and $j \in [m]$, pick $x_{i,j} \in \{0,1\}^n$ uniformly at random. Compute the value $h_i^t(x_{i,j})$. If $h_i^t(x_{i,j}) \neq \bot$, then store the value $(x_{i,j}, h_i^t(x_{i,j}))$ in a table $T_i$.

[a]Recall that $|A|$ is an integer parameter we will set, so this definition is not circular.
[b]If there is a tie, pick the lexicographically first $S$.
```

Finally, we state the inversion algorithm.

```
Inversion Algorithm for $f : \{0,1\}^n \to \{0,1\}^n$

Procedure Invert: Given $y \in \{0,1\}^n$,
  (1) If $y \in A$, then output a memorized preimage of $y$.
  (2) For all $i \in [\ell]$, set $u_i = g_i^\star(y)$.
  (3) For all $i \in [\ell]$ and $p \in \{0, \dots, t-1\}$, compute $h_i^p(u_i)$.
  (4) For all $i \in [\ell]$ and $p \in \{0, \dots, t-1\}$, check if $h_i^p(u_i)$ is in $T_i$ (i.e., $h_i^p(u_i) = h_i^t(x_{i,j})$ for some $j$). Let $F$ be the set given by
        $F = \{(i, j, p) : p \text{ is the least value satisfying } h_i^p(u_i) = h_i^t(x_{i,j})\}$.
      If $|F| \geq 10\ell$, then output fail and stop.
  (5) For all $(i, j, p) \in F$, if $f(h_i^{t-p-1}(x_{i,j})) = y$, then output $h_i^{t-p-1}(x_{i,j})$.
```

We note that there are several differences between the inversion procedure presented here, and the one presented in Fiat-Naor [17]:

1. Fiat-Naor construct A by sampling $|A|$ uniform random $x \in \{0,1\}^n$ and putting $(x, f(x))$ in A. This has the advantage of giving an efficient method for constructing A. We instead pick the "best possible" A and work with its corresponding N', as it simplifies the analysis.
2. Fiat-Naor use a different upper bound on j in the definition of $g_i^\star$. We use the upper bound $J = (n + \log t) \frac{2^n}{N'}$ to simplify the analysis and the circuit description.
3. Fiat-Naor add all the (i, j, p) for which the check passes to F (not just the triple with the *least* p). By only adding p with the "least value" property to F, as well as putting an upper bound on $|F|$, we simplify the running time analysis (and circuit description).

We show that the **Invert** procedure described above can be implemented by a circuit family that has a decent size bound in terms of the various parameters.

Theorem 3.3 (Circuit Upper Bound for Fiat-Naor). *Let $k \leq \ell \leq 2^n$, and suppose f has a circuit family of size $s(n)$. The procedure* **Invert** *on f can be implemented by a (randomized) circuit of size*

$$t \cdot |A| \cdot \mathsf{poly}(n) + t \cdot \ell \cdot J \cdot \mathsf{poly}(s) + \ell \cdot (t+m) \cdot \mathsf{poly}(n).$$

We prove Theorem 3.3 in Section 3.4.

Fiat-Naor [17] show that this procedure succeeds at inverting any given value with constant probability. Because our procedure is slightly different from the one in Fiat-Naor and for the sake of completeness, we sketch the proof of this in Section 3.5.

Theorem 3.4 (Fiat-Naor [17]). *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function. Assume*

- $\min\{t, m\} \geq 87$
- $k \geq 2t(n + \log t)\frac{2^n}{N'}$
- $t\ell m \geq N'$
- $|A| \geq 4t\frac{2^n}{N}m$
- $m \leq 2^n$

Then for every y in the range of f, the probability (over the randomness in the preprocessing step) that **Invert** *inverts f at y is $\Omega(1)$.*

Combining Theorem 3.3 and 3.4, we obtain a circuit construction that worst-case inverts f and beats exhaustive search.

Theorem 3.5. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function with a circuit of size $s(n) \geq n$. There is a circuit of size at most $2^{4n/5} \cdot \mathsf{poly}(s(n))$ that worst-case inverts f.*

Proof. (We assume all parameters are integers, and omit floors and ceilings for readability.) Set the parameter $|A| = 4 \cdot 2^{3n/5}$. This induces a set A and a value $N' := 2^n - |f^{-1}(A)|$. We divide into two cases depending on the size of N'.

Case 1: $N' \leq 2^{4n/5}$. In this case, there is a circuit of size $2^{4n/5} \cdot \mathsf{poly}(n)$ that implements table lookup on A in size $2^{3n/5} \cdot \mathsf{poly}(n)$, and directly stores inverses for all of the the remaining domain of size at most $N' \leq 2^{4n/5}$. In this case, these two table lookups cover all possible inputs to f.

Case 2: $N' > 2^{4n/5}$. Set $\ell := 2^{3n/5}$, $m := 2^{n/5}$, $t := \frac{N'}{2^{4n/5}} \leq 2^{n/5}$ (since $2^{4n/5} < N' \leq 2^n$, t can be rounded to a positive integer), and $k := 8tn \cdot \frac{2^n}{N'}$. Then

$$|A| = 4 \cdot 2^{3n/5} \geq 4\frac{N'}{2^{2n/5}} = 4 \cdot \left(\frac{N'}{2^{4n/5}}\right)^2 \cdot \frac{2^n}{N'} \cdot 2^{n/5} = 4t^2\frac{2^n}{N'}m,$$

and

$$t \cdot J = \frac{N'}{2^{4n/5}} \cdot (n + \log t)\frac{2^n}{N'} \leq 2n2^{n/5}.$$

Observe that this setting of the parameters satisfies all the constraints in the hypotheses of Theorem 3.3 and Theorem 3.4 when n is sufficiently large. Applying the bound of Theorem 3.3, there is a circuit for **Invert** of size at most

$$\begin{aligned}
&t \cdot |A| \cdot \mathsf{poly}(n) + \ell \cdot tJ \cdot \mathsf{poly}(s(n)) + \ell \cdot (t+m) \cdot \mathsf{poly}(n)\\
\leq\ & 2^{n/5}(4 \cdot 2^{3n/5}) \cdot \mathsf{poly}(n) + 2^{3n/5} \cdot (2n2^{n/5}) \cdot \mathsf{poly}(s(n))\\
&+ 2^{3n/5} \cdot (2^{n/5} + 2^{n/5}) \cdot \mathsf{poly}(n)\\
\leq\ & 2^{4n/5} \cdot \mathsf{poly}(s(n))
\end{aligned}$$

For any fixed y, this circuit will invert f on y with constant probability (over the randomness in the preprocessing step). Repeating this construction independently for $\mathsf{poly}(n)$ times and combining the resulting circuits yields a worst-case circuit for inverting f of size at most $2^{4n/5} \cdot \mathsf{poly}(s(n))$. □

3.4 Circuit Upper Bound: Proving Theorem 3.3

We now prove Theorem 3.3.

Proof of Theorem 3.3. The high-level idea is to carefully amortize the lookup table calls and function evaluations, so that they can all be done in batches. In this way, we can avoid the requirement of a random-access model, and can use circuits instead.

Each numbered step of **Invert** will correspond to some number of layers of our circuit. We will store the look-up tables A and $T_1, \ldots, T_\ell$ directly in the circuit, which takes $\mathsf{poly}(n) \cdot (|A| + \ell \cdot m)$ bits. These bits will be propagated to later layers of the circuit as they are needed (skipping layers when they are not needed). That way, we can always refer to the tables as needed throughout the computation of **Invert**.

Let us go through the steps of **Invert** one by one, and verify that they can be implemented with circuits of the desired size.

(1) Here, we only have to check whether y is in the lookup table A, for some $|A| \leq 2^n$. Applying Theorem 2.2, this can be done with a circuit of size $O(|A| \cdot \log^2(|A|)) \leq |A| \cdot \mathsf{poly}(n)$.

(2) Here, we have to evaluate $g_i^\star$ on y, for all $i \in [\ell]$. To implement $g_i^\star$, we will evaluate $g_i(y, j)$ on all relevant $j \in [J]$, and find the smallest j among these such that $f(g_i(y, j))$ is not in A. Each g_i is from a k-wise independent family, which Fiat-Naor implement as a degree-$(k-1)$ polynomial (in the variable z, say) over the finite field $\mathbb{F}_{N^2}$ where N is a power of two, and where the z^j coefficient of the polynomial is equal to $a_j \cdot i + b_j$. In this way, all g_i can be defined using the *same* polynomials. In particular, defining $P(z) = \sum_j a_j \cdot z^j$ and $Q(z) = \sum_j b_j \cdot z^j$, we have $g_i(z) = i \cdot P(z) + Q(z)$.
We have therefore reduced our evaluation problem to the following tasks:

(2a) Given two degree-$(k-1)$ polynomials $P(z)$ and $Q(z)$, we evaluate them on (y, j) for all $j \in [J]$.

(2b) For all $i \in [\ell]$, we evaluate $g_i(q) = i \cdot P(q) + Q(q)$ for all the J points q obtained in (a), and evaluate f on each $g_i(q)$.

(2c) Perform a batch lookup in table A on all $\ell \cdot J$ points $f(g_i(q))$ obtained in (b).

(2d) Determine for each $i \in [\ell]$ the minimum $j \in J$ such that $f(g_i(q)) \notin A$.

Applying Theorem 2.4, step (2a) can be done with arithmetic circuits of size

$$(k + J) \cdot \mathsf{poly}(\log(k + J)) \leq (k + J) \cdot \mathsf{poly}(n)$$

over the field; converting these circuits to Boolean circuits makes the size $\mathrm{poly}(n) \cdot (k+J) \cdot \mathrm{poly}(\log(k+J))$. Step (2b) takes $O(\ell \cdot J)$ arithmetic operations over the field followed by $O(\ell \cdot J)$ evaluations of a size-s circuit for f, translating to a circuit of size $\ell \cdot J \cdot \mathrm{poly}(s)$.

In Step (2c), we have a lookup table of $|A|$ points, and our batch is of size $\ell \cdot J$; applying Theorem 2.2, the lookups can be done in size

$$
\begin{aligned}
&O(n \cdot (|A| + \ell \cdot J) \cdot \log^2(n \cdot (|A| + \ell \cdot J))) \\
\leq &(|A| + \ell \cdot J) \cdot \mathrm{poly}(n).
\end{aligned}
$$

This produces ℓJ bits indicating which point is in the table or not. Finally, step (2d) just requires computing the first bit which is 1 among ℓ bit-vectors of length J, which can be done in size $O(\ell J)$. At the end of Step 2, the values $u_i = g_i^\star(y)$ for all $i \in [\ell]$ are stored in the circuit.

Under our hypothesis that $k \leq \ell$, the total size of Step 2 is at most

$$\ell \cdot J \cdot \mathrm{poly}(s) + (|A| + \ell \cdot J) \cdot \mathrm{poly}(n).$$

(3) In Step 3, we have to evaluate the function h_i on each u_i, for all $i \in [\ell]$. Then we have to compose $h_i(u_i)$ with itself for t times, storing the answer $h_i^p(u_i)$ for all $i \in [\ell]$ and $p \in \{0, \ldots, t-1\}$ for the next step.

Recall that $h_i(u_i) = g_i^\star(f(u_i))$. As in Step 2, we can perform this evaluation in a batch way: we evaluate $z_1 = f(u_1), \ldots, z_\ell = f(u_\ell)$ using a $\ell \cdot s(n)$ size circuit, then evaluate $g_i^\star$ on each u_i. This translates to evaluating

$$g_i(u_1, j) \ldots, g_i(u_\ell, j)$$

for all $j \in [J]$ and $i \in [\ell]$. As in Step 2, this amounts to J calls to multipoint evaluation of a degree-$(k-1)$ polynomial g on ℓ points, which can be done in size $(k+J) \cdot \mathrm{poly}(n)$, followed by ℓJ evaluations of f in size $\ell J \cdot \mathrm{poly}(s)$, followed by lookups into the table A in size $(|A| + \ell \cdot J) \cdot \mathrm{poly}(n)$. Indeed, the same argument shows that for *any* ℓ points $z_1, \ldots, z_\ell$ of our choice, we can evaluate $h_i(z_i)$ for all $i \in [\ell]$ using circuits of size

$$|A| \cdot \mathrm{poly}(n) + \ell \cdot J \cdot \mathrm{poly}(s).$$

Therefore we can compute the entire set of $\ell \cdot t$ points $P = \{h_i^p(u_i) \mid i \in [\ell], p \in \{0, \ldots, t-1\}\}$, using circuits of size

$$t \cdot |A| \cdot \mathrm{poly}(n) + t \cdot \ell \cdot J \cdot \mathrm{poly}(s).$$

(4) In Step 4, we first check for all $h_i^p(u_i)$ computed in the previous step whether or not $h_i^p(u_i) \in T_i$. Then we build the table F of triples (i, j, p), rejecting if F gets too large. That is, for all $i \in [\ell]$, we have to check whether the t strings $\left\{h_i^p(u_i) \,\middle|\, p \in \{0, \ldots, t-1\}\right\}$ appear in T_i. Furthermore, when such strings appear in T_i, we need to return a string $x_{i,j}$ associated with the string in table T_i, as well as the *least* value p such that $h_i^p(u_i) = h_i^t(x_{i,j})$. Abstractly, we need to solve the following task for ℓ query sets

$$Q_i = \left\{(p, h_i^p(u_i)) \,\middle|\, p \in \{0, \ldots, t-1\}\right\},$$

for all $i \in [\ell]$, paired with ℓ lists

$$L_i = \left\{(h_i^t(x_{i,j}), x_{i,j}) \,\middle|\, (x_{i,j}, h_i^t(x_{i,j})) \in T_i\right\},$$

and $N = n$:

Given queries $Q_i = \left\{(p_1, X_1'), \ldots, (p_t, X_t')\right\} \subseteq [2^N] \times \{0,1\}^N$, a list $L_i = \{(X_1, Y_1), \ldots, (X_m, Y_m)\} \subseteq \{0,1\}^N \times \{0,1\}^N$, and integer K, return up to K triples (p_j, X'_{p_j}, Y_j) such that $X'_{p_j} \in \{X_1, \ldots, X_m\}$ and p_j is minimal, if such triples exist.

In particular, we want to return up to 10ℓ triples over *all* ℓ batch queries (Q_i, L_i). This can be done by applying our circuits for batch queries with side information (Theorem 2.3) for ℓ times, maintaining a counter of the number of triples returned so far; this counter is passed from one batch query to the next, and the procedure is stopped early if the counter reaches 10ℓ. Implementing the batch lookups on all pairs Q_i, L_i requires size at most $\ell \cdot (t+m) \cdot \mathrm{poly}(n)$. In the end, our batch-lookup circuit returns $O(K \cdot n)$ bits encoding the relevant strings $x_{i,j} \in \{0,1\}^n$ along with their value $p \in \{0, \ldots, t-1\}$. If the batch-lookup circuit returns 10ℓ triples, the entire circuit rejects (as per step 4). Otherwise, the output of the circuit is treated as a representation of the set F.

(5) Finally, in step 5, $|F| < 10\ell$, and we have to iterate through each $(i, j, p) \in F$ in the lookup table results, and compute $f(h_i^{t-p-1}(x_{i,j}))$. We already have each of the relevant strings $x_{i,j}$ available from the output of our circuit in step 4, as well as the corresponding values $p \in \{0, \ldots, t-1\}$. In Step 3, we showed that for *any* ℓ points $z_1, \ldots, z_\ell$ of our choice, we can evaluate $h_i(z_i)$ for all $i \in [\ell]$ using a circuit of size

$$|A| \cdot \mathrm{poly}(n) + \ell \cdot J \cdot \mathrm{poly}(s).$$

Analogously, for any $c \leq 10\ell$ points, we can evaluate h_i on all of them using a circuit of asymptotically the same size. We can then compute h_i^p on the inputs $x_{i,j}$ for all $p \in \{0, \ldots, t-1\}$, in size

$$t \cdot |A| \cdot \mathrm{poly}(n) + t \cdot \ell \cdot J \cdot \mathrm{poly}(s).$$

Overall, the circuit size is dominated by Steps 3, 4, and 5; that is, the total size is at most

$$t \cdot |A| \cdot \mathrm{poly}(n) + t \cdot \ell \cdot J \cdot \mathrm{poly}(s) + \ell \cdot (t+m) \cdot \mathrm{poly}(n).$$

□

3.5 Analysis: Proving Theorem 3.4

Due to page requirements, we omit our review of the analysis of of Fiat-Naor [17]. A detailed review is included in the full version of our paper.

3.6 Proving Theorem 1.2

We conclude this section with the proof of Theorem 1.2.

Reminder of Theorem 1.2. *Let* Eval *denote a compression problem. There is a circuit family $\{C_{n,s}\}$ such that for all $n, s \in \mathbb{N}$, $C_{n,s}$ solves the compression problem for* Eval *on all strings of length n with descriptions of length at most s, and the size of $C_{n,s}$ is $2^{\frac{4}{5} \cdot s} \cdot \mathrm{poly}(n, s)$. Furthermore, $C_{n,s}(x)$ prints a description of length at most s for the input x of length n, whenever such a description exists.*

Proof of Theorem 1.2. Let $s, n \in \mathbb{N}$ and let Eval be polynomial-time computable, so that when restricted to inputs of length s and outputs of length n, Eval has a circuit $E_{n,s}$ of $\mathsf{poly}(n, s)$. Observe that for an n-bit input y, the problem of finding an x of length s such that $\mathrm{Eval}(x) = y$ is equivalent to solving the compression problem.

By Theorem 3.1, we can reduce the problem of inverting $E_{n,s} : \{0,1\}^s \to \{0,1\}^n$ to the problem of inverting another function $g : \{0,1\}^n \to \{0,1\}^n$ with $\mathsf{poly}(n)$-size circuits, in such a way that circuits of $S(n) \cdot \mathsf{poly}(n)$ size for inverting g imply circuits of $S(s+1) \cdot \mathsf{poly}(n)$ size for inverting $E_{n,s}$. Theorem 3.5 proves that there is a circuit that inverts g on all inputs, having size at most $2^{4n/5} \cdot \mathsf{poly}(n)$. Setting $S(n) = 2^{4n/5}$, we obtain circuits of size $2^{4s/5} \cdot \mathsf{poly}(n)$ for inverting $E_{n,s}$. □

4 CONSEQUENCES

We now turn to proving non-trivial circuit size bounds for MCSP, MINKT, and "compressible" NP relations, as mentioned in the Introduction.

4.1 Smaller Circuits for Finding Circuits (MCSP)

Reminder of Theorem 1.3. Search-MCSP$[s(n)]$ *on truth tables of length* 2^n *has circuits of size* $2^{\frac{4}{5} \cdot w + o(w)} \cdot \mathsf{poly}(2^n)$ *for all size functions* $s(n)$*, where* $w = s(n) \log_2(s(n) + n)$.

Proof. The idea is to define an appropriate evaluation function, and appeal to Theorem 1.2. In the case of MCSP, we want an Eval function that takes the encoding of a circuit as input, and outputs the circuit's truth table. However, in order to decisively beat the exhaustive search over $2^{O(s(n)\log(s(n)))}$ possible circuits, we need an essentially optimal encoding of circuits.

Let $s(n) \in [2^n]$ be our circuit size parameter (which we abbreviate as just s). Lemma 2.1 tells us there is a polynomial-time algorithm Enc such that, for every circuit C of size s and n inputs, there is some x of length $\ell_s := (1+o(1))s\log_2(s+n)$ such that $\mathrm{Enc}(x)$ outputs the description of a circuit C' of size s computing the same function as C. (Moreover, ℓ_s can be computed efficiently given s, although we do not need this property to construct a *non-uniform* circuit.) Let $\mathsf{TT}(C)$ be the function that takes the description of a circuit C with n inputs, and outputs its 2^n-bit truth table.

Define the evaluation function Eval to simply be $\mathrm{Eval}(x) := \mathsf{TT}(\mathrm{Enc}(x))$. Applying Theorem 1.2, there are circuits inverting the slice function $\mathrm{Eval}_{\ell_s, 2^n} : \{0,1\}^{\ell_s} \to \{0,1\}^{2^n}$ (Eval restricted to ℓ_s inputs and 2^n outputs) that have size

$$\begin{aligned} &2^{4\ell_s/5} \cdot \mathsf{poly}(\ell_s, 2^n) \\ &\leq 2^{\frac{4}{5} \cdot s(n)\log_2(s(n)+n) + o(s(n)\log_2(s(n)+n))} \cdot \mathsf{poly}(2^n). \end{aligned}$$

Given a truth table T of length 2^n, our final circuit inverts $\mathrm{Eval}_{\ell_s,2^n}$ on T. If inversion results in an x such that $\mathrm{Eval}_{\ell_s,2^n}(x) = T$, then our circuit outputs $\mathrm{Enc}(x)$ (an encoding of a circuit with truth table T), otherwise it outputs $\perp$ (failure). This completes the proof. □

4.2 Circuits for MINKT

Similarly, we can give nontrivial circuits for computing the K^t complexity of strings.

Reminder of Theorem 1.4. *For every time function* $t : \mathbb{N} \to \mathbb{N}$ *with* $t(n) \geq n$*, and parameters* $e, n \in \mathbb{N}$*, there is a circuit family that given any* n*-bit input* x*, outputs a program* y *of length at most* e *such that* y *prints the string* x *in at most* $t(n)$ *steps if such a* y *exists. The circuit family has size* $2^{\frac{4}{5}e} \cdot \mathsf{poly}(t(n))$.

Proof. Fix $e, n \in \mathbb{N}$, and set $t := t(n)$. Our Eval function simply takes an input d, treats d as a program, and runs d for t steps, outputting whatever string that d printed along the way. This Eval function, restricted to e-bit inputs and n-bit outputs, can be implemented by a circuit of size $\mathsf{poly}(t)$. Let $E_{e,n}$ be such a circuit.

Using the argument of Theorem 3.1, the problem of inverting $E_{e,n} : \{0,1\}^e \to \{0,1\}^n$ can be reduced to the problem of inverting another function $g : \{0,1\}^n \to \{0,1\}^n$ with $\mathsf{poly}(t, n)$-size circuits, such that circuits of $S(n) \cdot \mathsf{poly}(t, n)$ size for inverting g imply circuits of $S(e+1) \cdot \mathsf{poly}(t, n)$ size for inverting $E_{e,n}$. Theorem 3.5 proves that there is a circuit that inverts g on all inputs, having size at most $2^{4n/5} \cdot \mathsf{poly}(t, n)$. For $S(n) = 2^{4n/5}$, we obtain circuits of size $2^{4e/5} \cdot \mathsf{poly}(t(n))$ for inverting $E_{n,s}$. □

4.3 Circuits For Solving NP Relations on Compressible Instances

Here, we show how to solve the search problem for general NP relations faster when the entire instance-witness pair can be represented by a short program. In particular, our circuits are more efficient than enumerating over all short programs.

Let us recall the setup from the Introduction. Let $R \subseteq \{0,1\}^\star \times \{0,1\}^\star$ be any polynomial-time computable relation. For $n, p \in \mathbb{N}$, we define the task:

> **Compressible-**R**:** Given a string x of length n, if there is a program of size p which prints the pair (x, y) in $\mathsf{poly}(n)$ time such that $(x, y) \in R$, find a y' such that $(x, y') \in R$.

Enumerating all programs and checking them requires $2^p \cdot \mathsf{poly}(n)$ time; we show this "program-enumeration bottleneck" (as coined by [47]) can be circumvented using function inversion.

Reminder of Theorem 1.5. **Compressible-**R *can be solved by circuits of size* $2^{\frac{4}{5} \cdot p} \cdot \mathsf{poly}(n)$.

Proof. Let M be a machine deciding the polynomial-time relation R. Without loss of generality, let $k \in \mathbb{N}$ be some universal constant k such that $\ell(n) = kn^k$ is the length of witnesses for inputs of length n, i.e., $(x, y) \in R$ implies that $|y| = k|x|^k$.

Fix $p, n \in \mathbb{N}$. Define the function $\mathrm{Eval}_{p,n} : \{0,1\}^p \to \{0,1\}^{n+1}$ which takes in a string z of length p as input and treats z as a program, running z for $\mathsf{poly}(n)$ steps to obtain a string z', interpreted as a pair. If $M(z')$ accepts, and the first string x in the pair is n bits long, then the $(n+1)$-bit string $0x$ is output, otherwise $\mathrm{Eval}_{p,n}$ outputs 1^{n+1}.

It is easy to see that $\mathrm{Eval}_{p,n}$ can be implemented with $\mathsf{poly}(n)$-size circuits. Our circuit for **Compressible-**R on n-bit strings takes in an $x \in \{0,1\}^n$, and tries to invert $\mathrm{Eval}_{p,n}$ on the $(n+1)$-bit input $0x$. This inversion task can be accomplished with $2^{4p/5}$-size circuits, by Theorem 1.2, and the task exactly corresponds to finding a program z of length p which outputs $(x, y) \in R$ in $\mathrm{poly}(n)$ steps. This completes the proof. □

4.4 Computing Levin's Kt-Complexity on Average

Finally, in this section we construct a nontrivial circuit that computes MKtP on average. The idea is that a string x drawn from an efficiently computable distribution has small computational depth $\mathrm{cd}^t(x)$ with high probability, in which case we may assume that the time bound in $\mathrm{Kt}(x)$ is small. A similar idea was implicitly used in [37, 45] to characterize the existence of a one-way function by the average-case hardness of MKtP with respect to the uniform distribution. Here, we generalize the idea to any efficiently computable distribution.

We recall the notion of computational depth [3] and its properties. Recall that $\mathrm{K}^t(x)$ is defined to be the minimum length of a program d such that d prints x in t steps. The computational depth of a string measures the difference between time-bounded Kolmogorov complexity and plain Kolmogorov complexity. For a time bound $t \in \mathbb{N}$, the *computational depth* $\mathrm{cd}^t(x)$ of a string x is defined as

$$\mathrm{cd}^t(x) := \mathrm{K}^t(x) - \mathrm{K}(x).$$

Observe that $\mathrm{cd}^t(x)$ is never negative. First we show that, with high probability, $\mathrm{cd}^t(x)$ is small on random x drawn from an efficiently computable distribution.

Lemma 4.1. *For every function $t(n) \geq n$ and for every $t(n)$-time-computable distribution $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$ over $\{0,1\}^n$, it holds that*

$$\Pr_{x\sim\mathcal{D}_n}\left[\mathrm{cd}^{t'(n)}(x) \leq \ell\right] \geq 1 - n^{O(1)} \cdot 2^{-\ell}$$

for all $n, \ell \in \mathbb{N}$ and for $t'(n) = \mathrm{poly}(t(n))$.

Proof. Let $\mathcal{D}_n(x)$ denote the probability that x is sampled according to the distribution $\mathcal{D}_n$. By the coding theorem for computable distributions, for some $t'(n) = \mathrm{poly}(t(n))$, we have

$$\mathrm{K}^{t'(n)}(x) \leq -\log \mathcal{D}_n(x) + O(\log n)$$

for every $x \in \{0,1\}^n$ in the support of $\mathcal{D}_n$ and $n \in \mathbb{N}$ (see, e.g., [3, Theorem 3.5]). Thus,

$$\begin{aligned}\mathop{\mathbb{E}}_{x\sim\mathcal{D}_n}\left[2^{\mathrm{cd}^{t'(n)}(x)}\right] &\leq \mathop{\mathbb{E}}_{x\sim\mathcal{D}_n}\left[\frac{2^{-\mathrm{K}(x)}}{\mathcal{D}_n(x)} \cdot n^{O(1)}\right] \\ &= \sum_{x\in\{0,1\}^n} 2^{-\mathrm{K}(x)} \cdot n^{O(1)} \\ &\leq n^{O(1)},\end{aligned}$$

where the last inequality holds due to Kraft's inequality and its relation with plain Kolmogorov complexity [35]. Therefore we have

$$\Pr_{x\sim\mathcal{D}_n}\left[\mathrm{cd}^{t'(n)}(x) \geq \ell\right] = \Pr_{x\sim\mathcal{D}_n}\left[2^{\mathrm{cd}^{t'(n)}(x)} \geq 2^\ell\right] \leq n^{O(1)} \cdot 2^{-\ell},$$

by Markov's inequality. The lemma follows. □

Next, we introduce a variant of Kt-complexity in which we impose a time upper bound.

Definition 4.2. *For a string $x \in \{0,1\}^*$ and a time bound $T \in \mathbb{N}$, define*

$$\mathrm{Kt}^{\leq T}(x) := \min\{|d| + \log t \mid U(d) \text{ outputs } x \text{ in time at most } T\},$$

where U is an efficient universal Turing machine.

Lemma 4.3. *If $\mathrm{cd}^T(x) \leq \ell$, then $\mathrm{Kt}(x) = \mathrm{Kt}^{\leq 2^\ell T}(x)$.*

Proof. It suffices to prove that $\mathrm{Kt}(x) \geq \mathrm{Kt}^{\leq 2^\ell T}(x)$. Let d and $t \in \mathbb{N}$ be a program and a time bound, respectively, such that $\mathrm{Kt}(x) = |d| + \log t$ and U outputs x on input d in time t. Our goal is to prove $t \leq 2^\ell T$.

Since $U(d) = x$, we have $|d| \geq \mathrm{K}(x) \geq \mathrm{K}^T(x) - \ell$, and thus $\mathrm{Kt}(x) \geq \mathrm{K}^T(x) - \ell + \log t$. By the definition of $\mathrm{Kt}(x)$, we also have $\mathrm{Kt}(x) \leq \mathrm{K}^T(x) + \log T$. Combining these two inequalities, we obtain $\mathrm{K}^T(x) - \ell + \log t \leq \mathrm{Kt}(x) \leq \mathrm{K}^T(x) + \log T$, which implies that $t \leq 2^\ell T$. □

We are ready to prove Theorem 1.6.

Reminder of Theorem 1.6. *For all functions $s(n)$ and $t(n) \geq n$, there exists a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ of size $2^{\frac{4}{5}s(n)} \cdot \mathrm{poly}(t(n))$ such that for any $t(n)$-time-computable distribution $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$ over $\{0,1\}^n$, for all large $n \in \mathbb{N}$, with probability at least $1 - \frac{1}{t(n)}$ over a random input x drawn from $\mathcal{D}_n$, on input x, the circuit C_n outputs a program y and $t \in \mathbb{N}$ such that y prints the string x in t steps and $|y| + \log t \leq s(n)$ if $\mathrm{Kt}(x) \leq s(n)$.*

Proof of Theorem 1.6. We construct a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ that computes a pair (y, t) that witnesses $\mathrm{Kt}^{\leq t'(n)}(x) \leq s(n)$ for some $t'(n) = \mathrm{poly}(t(n))$ to be chosen.

The circuit C_n can be constructed by using the circuit of Theorem 1.4 because

$$\mathrm{Kt}^{\leq t'(n)}(x) = \min\left\{\mathrm{K}^t(x) + \log t \;\middle|\; t \leq t'(n)\right\}.$$

Thus, the size of C_n is at most $2^{\frac{4}{5}s(n)} \cdot \mathrm{poly}(t(n))$.

It remains to claim that C_n solves the search version of MKtP on average with probability $1 - 1/t(n)$. Let $\mathcal{D}_n$ be a $t(n)$-time-computable distribution. Then, by Lemma 4.1, for large enough $t_1(n) = \mathrm{poly}(t(n))$ we have $\mathrm{cd}^{t_1(n)}(x) \leq \log t_1(n)$ with probability at least $1 - 1/t(n)$ over $x \sim \mathcal{D}_n$. By Lemma 4.3, for any x with $\mathrm{cd}^{t_1(n)}(x) \leq \log t_1(n)$, we have $\mathrm{Kt}^{\leq t_1(n)^2}(x) = \mathrm{Kt}(x)$. Letting $t'(n) := t_1(n)^2 = \mathrm{poly}(t(n))$, we obtain that C_n computes a witness for $\mathrm{Kt}(x) \leq s(n)$ with probability $1 - 1/t(n)$ over $x \sim \mathcal{D}_n$. □

5 CONCLUSION

In this paper, we have shown how function inversion can be applied to make a dent in longstanding open problems in computational complexity, such as the circuit complexity of MCSP. In particular, the old "perebor conjecture" (see [46]) that brute-force is required for compression problems, is refuted when we are allowed non-uniform circuits as our algorithmic model. Our work raises a variety of new questions.

- The first obvious open question is whether there are smaller circuits than $2^{4e/5} \cdot \mathrm{poly}(n)$ for finding compressed descriptions of length e. There seems to be a barrier to finding circuits significantly smaller than $2^{e/2} \cdot \mathrm{poly}(n)$: if they existed, we could solve NP problems such that the witness is of length equal to the input, with circuits that are smaller than $2^n \cdot \mathrm{poly}(n)$ (see Section 1.2). In the black-box setting, there are $\Omega(2^{n/2})$ lower bounds on function inversion via data structures: for example, De et al. [15] show a time-space tradeoff lower bound of $T \cdot S \geq \Omega(\varepsilon 2^n)$ for inverting a function on at least an ε-fraction of inputs with a space-S data structure that makes T function queries, generalizing earlier work of Yao [51].

- Is there a *randomized algorithm* for finding descriptions of length ℓ for strings of length n, that runs in time $2^{\alpha\ell} \cdot \mathrm{poly}(n)$ for some $\alpha < 1$? We do not achieve this, because our lookup tables can take a long time to construct. If we could find a randomized algorithm, then by results of Chen and Tell [11], there would also be a *deterministic algorithm* running in time $2^{\beta\ell} \cdot \mathrm{poly}(n)$ for some $\beta < 1$, assuming reasonable complexity hypotheses. In such a case, we would have a more "decisive" refutation of the old conjecture that MCSP and other compression problems require exhaustive search.
- We have shown how time-bounded Kolmogorov complexity can be determined faster than trying all programs up to a given size. There are other time-bounded versions of Kolmogorov complexity which appear even harder to compute, such as MKtP. We showed that MKtP can be solved *on average* with nontrivial circuits (Theorem 1.6); roughly speaking, this is because in the average case, the complexity of MKtP behaves similarly to NP [37, 45]. MKtP is known to be complete for EXP (exponential time) under various reduction types [1]. Could there be non-trivial circuits for MKtP in the *worst case*, as well? This question seems to be gently poking at the well-known hypothesis in derandomization that $\mathsf{TIME}[2^n]$ requires exponential-size circuits [33].
- We know that $\mathsf{UP} \cap \mathsf{coUP} \neq \mathsf{P}$ iff there exists a bijective worst-case one-way function [29]. Given the results of this paper, are there nontrivial circuits for problems in $\mathsf{UP} \cap \mathsf{coUP}$?
- Pairwise-independent hashing was crucial in order to achieve a circuit size bound for compression that is exponential only in the encoding length, which is a "small" parameter relative to the input. Could a similar hashing method be more broadly useful in developing new parameterized algorithms (or circuits)? Certainly hashing methods are already widely applied in parameterized algorithms (probably the most famous one is *color coding* [2]) but our trick seems somewhat different.

ACKNOWLEDGMENTS

We thank Yuval Ishai, Moni Naor, Hanlin Ren, and Rahul Santhanam for discussions. This work was partially completed while the authors were visiting the Simons Institute for the Theory of Computing in Spring 2023.

REFERENCES

[1] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. 2006. Power from Random Strings. *SIAM J. Comput.* 35, 6 (2006), 1467–1493. https://doi.org/10.1137/050628994
[2] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-Coding. *J. ACM* 42, 4 (1995), 844–856. https://doi.org/10.1145/210332.210337
[3] Luis Antunes, Lance Fortnow, Dieter van Melkebeek, and N. V. Vinodchandran. 2006. Computational depth: Concept and applications. *Theor. Comput. Sci.* 354, 3 (2006), 391–404. https://doi.org/10.1016/j.tcs.2005.11.033
[4] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity - A Modern Approach.* Cambridge University Press.
[5] Daniel J. Bernstein and Tanja Lange. 2013. Non-uniform Cracks in the Concrete: The Power of Free Precomputation. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT).* 321–340. https://doi.org/10.1007/978-3-642-42045-0_17
[6] Andrej Bogdanov and Luca Trevisan. 2006. Average-Case Complexity. *Foundations and Trends in Theoretical Computer Science* 2, 1 (2006). https://doi.org/10.1561/0400000004
[7] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. 2016. Learning Algorithms from Natural Proofs. In *Proceedings of the Conference on Computational Complexity (CCC).* 10:1–10:24. https://doi.org/10.4230/LIPIcs.CCC.2016.10
[8] Dror Chawin, Iftach Haitner, and Noam Mazor. 2020. Lower Bounds on the Time/Memory Tradeoff of Function Inversion. In *Proceedings of the Theory of Cryptography Conference (TCC).* 305–334. https://doi.org/10.1007/978-3-030-64381-2_11
[9] Lijie Chen, Shuichi Hirahara, Igor Carboni Oliveira, Ján Pich, Ninad Rajgopal, and Rahul Santhanam. 2022. Beyond Natural Proofs: Hardness Magnification and Locality. *J. ACM* 69, 4 (2022), 25:1–25:49. https://doi.org/10.1145/3538391
[10] Lijie Chen, Ce Jin, and R. Ryan Williams. 2019. Hardness Magnification for all Sparse NP Languages. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS).* 1240–1255. https://doi.org/10.1109/FOCS.2019.00077
[11] Lijie Chen and Roei Tell. 2020. Simple and fast derandomization from very hard functions: Eliminating randomness at almost no cost. *Electron. Colloquium Comput. Complex.* 27 (2020), 148.
[12] Ruiwen Chen and Valentine Kabanets. 2016. Correlation bounds and #SAT algorithms for small linear-size circuits. *Theor. Comput. Sci.* 654 (2016), 2–10. https://doi.org/10.1016/J.TCS.2016.05.005
[13] Mahdi Cheraghchi, Shuichi Hirahara, Dimitrios Myrisiotis, and Yuichi Yoshida. 2021. One-Tape Turing Machine and Branching Program Lower Bounds for MCSP. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS).* 23:1–23:19. https://doi.org/10.4230/LIPIcs.STACS.2021.23
[14] Henry Corrigan-Gibbs and Dmitry Kogan. 2019. The Function-Inversion Problem: Barriers and Opportunities. In *Proceedings of the Theory of Cryptography Conference (TCC).* 393–421. https://doi.org/10.1007/978-3-030-36030-6_16
[15] Anindya De, Luca Trevisan, and Madhur Tulsiani. 2010. Time Space Tradeoffs for Attacks against One-Way Functions and PRGs. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings.* 649–665. https://doi.org/10.1007/978-3-642-14623-7_35
[16] Pavel Dvořák, Michal Koucký, Karel Král, and Veronika Slívová. 2021. Data Structures Lower Bounds and Popular Conjectures. In *Proceedings of the European Symposium on Algorithms (ESA).* 39:1–39:15. https://doi.org/10.4230/LIPICS.ESA.2021.39
[17] Amos Fiat and Moni Naor. 1999. Rigorous Time/Space Trade-offs for Inverting Functions. *SIAM J. Comput.* 29, 3 (1999), 790–803. https://doi.org/10.1137/S0097539795280512
[18] Charles M. Fiduccia. 1972. Polynomial Evaluation via the Division Algorithm: The Fast Fourier Transform Revisited. In *Proceedings of the Symposium on Theory of Computing (STOC).* 88–93. https://doi.org/10.1145/800152.804900
[19] Gudmund Skovbjerg Frandsen and Peter Bro Miltersen. 2005. Reviewing bounds on the circuit size of the hardest functions. *Inf. Process. Lett.* 95, 2 (2005), 354–357. https://doi.org/10.1016/j.ipl.2005.03.009
[20] Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan. 2020. Data structures meet cryptography: 3SUM with preprocessing. In *Proceedings of the Symposium on Theory of Computing (STOC).* 294–307. https://doi.org/10.1145/3357713.3384342
[21] Alexander Golovnev, Siyao Guo, Spencer Peters, and Noah Stephens-Davidowitz. 2023. Revisiting Time-Space Tradeoffs for Function Inversion. In *Proceedings of the International Cryptology Conference (CRYPTO).* 453–481. https://doi.org/10.1007/978-3-031-38545-2_15
[22] Alexander Golovnev, Alexander S. Kulikov, Alexander V. Smal, and Suguru Tamaki. 2016. Circuit Size Lower Bounds and #SAT Upper Bounds Through a General Framework. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science (MFCS).* 45:1–45:16. https://doi.org/10.4230/LIPICS.MFCS.2016.45
[23] Martin E. Hellman. 1980. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory* 26, 4 (1980), 401–406. https://doi.org/10.1109/TIT.1980.1056220
[24] Shuichi Hirahara. 2018. Non-Black-Box Worst-Case to Average-Case Reductions within NP. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS).* 247–258. https://doi.org/10.1109/FOCS.2018.00032
[25] Shuichi Hirahara. 2020. Non-Disjoint Promise Problems from Meta-Computational View of Pseudorandom Generator Constructions. In *Proceedings of the Computational Complexity Conference (CCC).* 20:1–20:47. https://doi.org/10.4230/LIPIcs.CCC.2020.20
[26] Shuichi Hirahara. 2021. Average-case hardness of NP from exponential worst-case hardness assumptions. In *Proceedings of the Symposium on Theory of Computing*

(STOC). 292–302. https://doi.org/10.1145/3406325.3451065
[27] Shuichi Hirahara. 2022. NP-Hardness of Learning Programs and Partial MCSP. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 968–979. https://doi.org/10.1109/FOCS54457.2022.00095
[28] Shuichi Hirahara and Mikito Nanashima. 2021. On Worst-Case Learning in Relativized Heuristica. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 751–758. https://doi.org/10.1109/FOCS52979.2021.00078
[29] Christopher M. Homan and Mayur Thakur. 2003. One-way permutations and self-witnessing languages. *J. Comput. Syst. Sci.* 67, 3 (2003), 608–622. https://doi.org/10.1016/S0022-0000(03)00068-0
[30] Rahul Ilango. 2023. SAT Reduces to the Minimum Circuit Size Problem with a Random Oracle. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 733–742. https://doi.org/10.1109/FOCS57990.2023.00048
[31] Rahul Ilango, Hanlin Ren, and Rahul Santhanam. 2022. Robustness of average-case meta-complexity via pseudorandomness. In *Proceedings of the Symposium on Theory of Computing (STOC)*. 1575–1583. https://doi.org/10.1145/3519935.3520051
[32] Russell Impagliazzo and Leonid A. Levin. 1990. No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 812–821. https://doi.org/10.1109/FSCS.1990.89604
[33] Russell Impagliazzo and Avi Wigderson. 1997. P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma. In *Proceedings of the Symposium on the Theory of Computing (STOC)*. 220–229. https://doi.org/10.1145/258533.258590
[34] Leonid A. Levin. 1986. Average Case Complete Problems. *SIAM J. Comput.* 15, 1 (1986), 285–286. https://doi.org/10.1137/0215020
[35] Ming Li and Paul M. B. Vitányi. 2019. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition*. Springer. https://doi.org/10.1007/978-3-030-11298-1
[36] Yanyi Liu and Rafael Pass. 2020. On One-way Functions and Kolmogorov Complexity. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 1243–1254.
[37] Yanyi Liu and Rafael Pass. 2021. On the Possibility of Basing Cryptography on EXP≠ BPP. In *Proceedings of the International Cryptology Conference (CRYPTO)*. 11–40. https://doi.org/10.1007/978-3-030-84242-0_2
[38] Noam Mazor and Rafael Pass. 2024. The Non-Uniform Perebor Conjecture for Time-Bounded Kolmogorov Complexity Is False. In *Proceedings of the Innovations in Theoretical Computer Science Conference (ITCS)*. 80:1–80:20. https://doi.org/10.4230/LIPICS.ITCS.2024.80
[39] Dylan M. McKay, Cody D. Murray, and R. Ryan Williams. 2019. Weak lower bounds on resource-bounded compression imply strong separations of complexity classes. In *Proceedings of the Symposium on Theory of Computing (STOC)*. 1215–1225. https://doi.org/10.1145/3313276.3316396
[40] Igor Carboni Oliveira, Ján Pich, and Rahul Santhanam. 2019. Hardness Magnification near State-Of-The-Art Lower Bounds. In *Proceedings of the Computational Complexity Conference (CCC)*. 27:1–27:29. https://doi.org/10.4230/LIPIcs.CCC.2019.27
[41] Igor Carboni Oliveira and Rahul Santhanam. 2018. Hardness Magnification for Natural Problems. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 65–76.
[42] Wolfgang J. Paul. 1976. Realizing Boolean Functions on Disjoint sets of Variables. *Theor. Comput. Sci.* 2, 3 (1976), 383–396. https://doi.org/10.1016/0304-3975(76)90089-X
[43] Nicholas Pippenger and Michael J. Fischer. 1979. Relations Among Complexity Measures. *J. ACM* 26, 2 (1979), 361–381. https://doi.org/10.1145/322123.322138
[44] Alexander A. Razborov and Steven Rudich. 1997. Natural Proofs. *J. Comput. Syst. Sci.* 55, 1 (1997), 24–35. https://doi.org/10.1006/jcss.1997.1494
[45] Hanlin Ren and Rahul Santhanam. 2021. Hardness of KT Characterizes Parallel Cryptography. In *Proceedings of the Computational Complexity Conference (CCC)*. 35:1–35:58. https://doi.org/10.4230/LIPIcs.CCC.2021.35
[46] Boris A. Trakhtenbrot. 1984. A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms. *IEEE Annals of the History of Computing* 6, 4 (1984), 384–400. https://doi.org/10.1109/MAHC.1984.10036
[47] Luca Trevisan. 2010. The Program-Enumeration Bottleneck in Average-Case Complexity Theory. In *Proceedings of the Conference on Computational Complexity (CCC)*. 88–95. https://doi.org/10.1109/CCC.2010.18
[48] Leslie G. Valiant and Vijay V. Vazirani. 1986. NP is as Easy as Detecting Unique Solutions. *Theor. Comput. Sci.* 47, 3 (1986), 85–93. https://doi.org/10.1016/0304-3975(86)90135-0
[49] Joachim von zur Gathen and Jürgen Gerhard. 2013. *Modern Computer Algebra (3. ed.)*. Cambridge University Press.
[50] Ryan Williams. 2013. Improving Exhaustive Search Implies Superpolynomial Lower Bounds. *SIAM J. Comput.* 42, 3 (2013), 1218–1244. https://doi.org/10.1137/10080703X
[51] Andrew Chi-Chih Yao. 1990. Coherent Functions and Program Checkers (Extended Abstract). In *Proceedings of the Symposium on Theory of Computing (STOC)*. 84–94. https://doi.org/10.1145/100216.100226

Received 13-NOV-2023; accepted 2024-02-11