Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 156 (2023)

Symmetric Exponential Time Requires Near-Maximum Circuit Size: Simplified, Truly Uniform

Zeyong Li*

Abstract

In a recent breakthrough, Chen, Hirahara and Ren [CHR24] prove that $\mathsf{S_2E}/_1 \not\subset \mathsf{SIZE}[2^n/n]$ by giving a single-valued $\mathsf{FS_2P}$ algorithm for the Range Avoidance Problem (Avoid) that works for infinitely many input size n.

Building on their work, we present a simple single-valued $\mathsf{FS_2P}$ algorithm for Avoid that works for all input size n. As a result, we obtain the circuit lower bound $\mathsf{S_2E} \not\subset$ *i.o.*-$\mathsf{SIZE}[2^n/n]$ and many other corollaries:

1. Almost-everywhere near-maximum circuit lower bound for $\mathsf{\Sigma_2E} \cap \mathsf{\Pi_2E}$ and $\mathsf{ZPE^{NP}}$.
2. Pseudodeterministic $\mathsf{FZPP^{NP}}$ constructions for combinatorial objects such as: Ramsey graphs, rigid matrices, pseudorandom generators, two-source extractors, linear codes, hard truth tables, and K^{poly}-random strings.

1 Introduction

Proving circuit lower bounds has been one of the most fundamental problems in complexity theory, and has close connections to many other fundamental questions such as P versus NP, derandomization and so on. For instance, if we could show $\mathsf{E} \not\subset \mathsf{SIZE}[2^{o(n)}]$, then we achieve unconditional derandomization, i.e. $\mathsf{prBPP} = \mathsf{prP}$ [NW94, IW97]. Morally speaking, it is a quest to distinguish the computational power between uniform and non-uniform computations.

In the search of exponential circuit lower bound, we know that almost all n-bit boolean functions requires near-maximum $(2^n/n)$-sized circuit via a simple counting argument [Sha49, FM05]. While such argument is inherently non-constructive, it serves as some form of evidence that we should be optimistic about finding one such boolean function in some not-too-large complexity class.

However, limited progress has been made over the past few decades in the search of a small complexity class with exponential circuit lower bound. In 1982, Kannan [Kan82] showed that $\mathsf{\Sigma_3E} \cap \mathsf{\Pi_3E}$ contains a language with maximum circuit complexity. The frontier was later pushed to $\mathsf{\Delta_3E} = \mathsf{E^{\Sigma_2P}}$ by Miltersen, Vinodchandran and Watanabe [MVW99], which persisted to be the state of the art for more than twenty years.

Very recently in a breakthrough result, Chen, Hirahara and Ren [CHR24] prove that:

$$\mathsf{S_2E}/_1 \not\subset \mathsf{SIZE}[2^n/n] \ .$$

That is, the symmetric time class $\mathsf{S_2E}$ with one bit of advice requires near-maximum circuit complexity. Despite requiring one bit of advice, this is a huge improvement compared to the previous result since $\mathsf{S_2E} \subseteq \mathsf{ZPE^{NP}} \subseteq \mathsf{\Sigma_2E}$ is expected to be a much smaller class compared to $\mathsf{\Delta_3E} = \mathsf{E^{\Sigma_2P}}$, assuming that the exponential hierarchy does not collapse. And it turns out that this exciting result is indeed a corollary of a *single-valued* algorithm for the Range Avoidance problem (Avoid).

*National University of Singapore. Email: li.zeyong@u.nus.edu

ISSN 1433-8092

The Range Avoidance Problem In the past few years, study on the range avoidance problem has been another exciting line of research [KKMP21, Kor21, GLW22, RSW22, CHLR23, GGNS23, ILW23]. The problem itself is defined as follows: given an expanding circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$, find a string not in the image of C. That is, output $y \in \{0,1\}^{n+1}$ where $\forall x \in \{0,1\}^n, C(x) \neq y$.

At a first glance, Avoid might seem to be a easy problem: a random string y would be a non-image of C with probability at least $1/2$. However, it is unclear how to amplify the probability of success without an NP oracle. And given an NP oracle, Avoid can be trivially solved in $\mathsf{FZPP}^{\mathsf{NP}}$. Somewhat surprisingly, Korten [Kor21] showed that Avoid is as hard as finding optimal explicit constructions of important combinatorial objects such as Ramsey graphs [Rad21], rigid matrices [GLW22, GGNS23], pseudorandom generators [CT22], two-source extractors [CZ19, Li23], linear codes [GLW22], hard truth tables [Kor21], and strings with maximum time-bounded Kolmogorov complexity (K^{poly}-random strings) [RSW22]. Therefore, finding any non-trivial algorithm for Avoid implies algorithms for constructing these important objects.

Single-Valued Algorithm Let Π be a search problem where Π_x denotes the set of solutions for input x. Morally speaking, A single-valued algorithm A on input x succeeds only when it outputs some canonical solution $y_x \in \Pi_x$. Here are two examples:

- A single-valued FNP algorithm should have at least one successful computational path and should output $\perp$ in all other computational paths. (Studied as NPSV constructions in, e.g. [HNOS96]).
- A single-valued FZPP algorithm should succeed on most (e.g. $\geq 2/3$ fraction) computational paths and output $\perp$ otherwise. (Studied as pseudodeterministic constructions in, e.g. [GG11]).

In particular, the trivial $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm for Avoid (i.e. sample a string and check with the NP oracle) is inherently not single-valued: the outputs would be different in almost all executions!

As pointed out in [CHR24] and many other previous works, circuit lower bounds can be viewed as single-valued construction of hard truth tables. In particular, if for all input size n, one could compute (consistently the same) truth table that is hard against all $s(n)$-size circuits, then the language whose characteristic function is set to the truth table, would be a hard language $\notin \mathsf{SIZE}[s(n)]$. Given that finding hard truth table reduces to Avoid, this connects the two tasks: proving circuit lower bound and finding single-valued algorithm for Avoid.

1.1 Our Results

1.1.1 Algorithm for the Range Avoidance Problem

[CHR24] presented a single-valued $\mathsf{FS_2P}$ algorithm for Avoid that works *infinitely often*. I.e., for infinitely many (but unknown) input size n, there is a $\mathsf{S_2P}$ machine[1], on input a circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$, outputs a canonical non-image of C in the successful computational paths and $\perp$ otherwise.

In this work, we extend their algorithm to a single-valued $\mathsf{FS_2P}$ algorithm for Avoid that works *for all input size n*.

Theorem 1.1. *There is a single-valued $\mathsf{FS_2P}$ algorithm A: when given any circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$ as input, $A(C)$ outputs y_C such that $y_C \notin \mathrm{Im}(C)$.*

The remaining results are corollaries of our new single-valued $\mathsf{FS_2P}$ algorithm for Avoid.

1.1.2 Almost-everywhere near-maximum circuit lower bounds

In [CHR24], their circuit lower bound requires one bit of advice. This is because their $\mathsf{FS_2P}$ algorithm for Avoid only works for infinitely many input size n, and the one bit of advice is necessary for indicating which input size is 'good'. As our algorithm works for all input size n, the circuit lower bound that we obtain are almost-everywhere and completely removes any non-uniform advice:

[1]To provide some intuition for readers who are unfamiliar with this class, $\mathsf{S_2P}$ is contained in $\mathsf{ZPP}^{\mathsf{NP}}$[Cai07].

Theorem 1.2. $\mathsf{S_2E} \not\subset$ *i.o.*-$\mathsf{SIZE}[2^n/n]$. *Moreover, this holds in every relativized world.*

Similar to [CHR24], our results fully relativize.

Via known results where $\mathsf{S_2E} \subseteq \mathsf{ZPE}^{\mathsf{NP}}$ [Cai07] and $\mathsf{ZPE}^{\mathsf{NP}} \subseteq \mathsf{\Sigma_2E} \cap \mathsf{\Pi_2E}$, we obtain the following corollaries:

Corollary 1.3. $\mathsf{ZPE}^{\mathsf{NP}} \not\subset$ *i.o.*-$\mathsf{SIZE}[2^n/n]$. *Moreover, this holds in every relativized world.*

Corollary 1.4. $\mathsf{\Sigma_2E} \cap \mathsf{\Pi_2E} \not\subset$ *i.o.*-$\mathsf{SIZE}[2^n/n]$. *Moreover, this holds in every relativized world.*

1.1.3 Explicit constructions

Next, by Cai's theorem [Cai07], we have $\mathsf{S_2P} \subseteq \mathsf{ZPP}^{\mathsf{NP}}$. Hence, Avoid and all explicit construction problems admit a single-valued $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm. Equivalently speaking, Avoid and all explicit construction problems admit a pseudodeterministic (with an NP oracle) algorithm, where a pseudodeterministic algorithm for a search problem is a probabilistic algorithm that with high probability outputs a fixed solution on any given input.

Theorem 1.5. *There is a single-valued* $\mathsf{FZPP}^{\mathsf{NP}}$ *algorithm A: when given any circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$ as input, $A(C)$ outputs y_C with probability at least $2/3$ and $y_C \notin \mathrm{Im}(C)$.*

Corollary 1.6 (Informal)**.** *There are zero-error pseudodeterministic constructions for the following objects with an* NP *oracle for every input size n: Ramsey graphs, rigid matrices, pseudorandom generators, two-source extractors, linear codes, hard truth tables, and K^{poly}-random strings.*

1.1.4 Missing-String problem

Lastly, we point out a connection to the MissingString problem. The MissingString problem is defined as follows: given a list of m strings $x_1, \dots, x_m \in \{0,1\}^n$ where $m < 2^n$, the goal is to output any string $y \in \{0,1\}^n$ not in the list.

In [VW23], Vyas and Williams connected the circuit complexity of the MissingString with the (relativized) circuit complexity of $\mathsf{\Sigma_2E}$.

Theorem 1.7 ([VW23])**.** *The following are equivalent:*

1. $\mathsf{\Sigma_2E}^A \not\subset$ *i.o.*-$\mathsf{SIZE}^A[2^{\Omega(n)}]$ *for every oracle A;*
2. *for $m = 2^{\Omega(n)}$, the* MissingString *problem can be solved by a uniform family of size-$2^{\mathrm{poly}(n)}$ depth-3* AC^0 *circuits.*

As a corollary of our circuit lower bound which relativizes, we can conclusively claim that:

Corollary 1.8. *For $m = 2^{\Omega(n)}$, the* MissingString *problem can be solved by a uniform family of size-$2^{\mathrm{poly}(n)}$ depth-3* AC^0 *circuits.*

1.2 Proof Overview

1.2.1 Korten's Reduction

We start by reviewing Korten's reduction from [Kor21], which reduces Avoid on a circuit with n-bit stretch (i.e. maps n-bit input to $2n$-bit output) to Avoid on some other circuit with much longer stretch.

Given any circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$ and parameter $T = n \cdot 2^k$, Korten builds another circuit $\mathsf{GGM}_T[C] : \{0,1\}^n \to \{0,1\}^T$ by applying the circuit C in a perfect binary tree:

1. Assign the root vertex $(0,0)$ with value $v_{0,0} = x$. Build a perfect binary tree of height k where (i,j) denotes the jth vertex on the ith level for $0 \le i \le k$ and $0 \le j \le 2^i - 1$.
2. For each vertex (i,j) on the tree, evaluate $y = C(v_{i,j})$ and assign its left child with the first n bits of y and its right child with the last n bits of y.

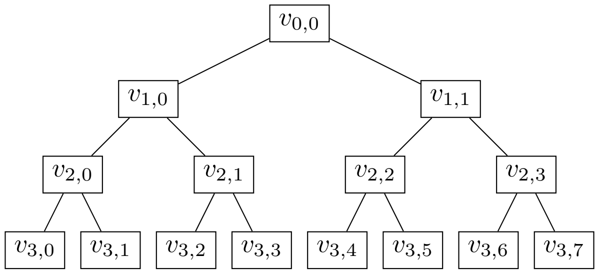

Figure 1: An illustration of a GGM tree of height 3.

3. The output of $\mathsf{GGM}_T[C](x)$ is simply the concatenation of the values of the 2^k leaves.

Notice that on any fixed input $x \in \{0,1\}^n$, every vertex on the GGM tree has an n-bit value. Hence, we call it a *fully-assigned* GGM tree. It is not hard to see that one can efficiently (takes time linear in the height of the tree, k) evaluate the assigned value at any vertex by traversing the tree and apply the circuit C at most k times. In other words, a *fully-assigned* GGM tree has small circuit complexity.

Korten's reduction asserts that given an NP oracle and any $f \in \{0,1\}^T \setminus \mathrm{Im}(\mathsf{GGM}_T[C])$, there is a deterministic algorithm that finds a non-image of C and runs in time $\mathrm{poly}(T, n)$. And the algorithm is simple:

1. Set the assigned values of the leaves to be f.
2. Next, traverse the tree in a simple bottom up manner. I.e. traverse the 2^{k-1} vertices on the $(k-1)$th level one by one (say, from right to left), then proceed to the $(k-2)$th level and so on, until reaching root.
3. For each interval vertex u traversed, assign v_u with the lexicographically first[2] n-bit string x such that $C(x)$ correctly evaluates to the assigned values of u's children (Note that this step uses the NP oracle).
4. Whenever such string cannot be found, we successfully find a non-image of C (i.e. the assigned values of u's children). The algorithm now returns with the non-image of C found and assigns $\perp$ to all remaining vertices.

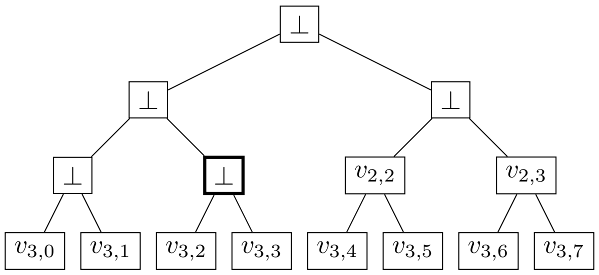

Figure 2: An illustration of the *partially-assigned* GGM tree from running Korten's algorithm.

[2]the original reduction from [Kor21] didn't specify the lexicographically first requirement. This requirement is important for [CHR24] and us in order to obtain a single-valued algorithm, and it comes for free given the NP oracle.

1.2.2 History of the Reduction

The computational history of Korten's reduction on fixed input (C, f) is fully characterised by a *partially-assigned* GGM tree i.e. some of the vertices are assigned $\perp$. We are interested in the computaional history because it has a few nice properties:

1. **Contains a Canonical Solution:** Notice that the execution of Korten's algorithm is fully deterministic. Hence, it produces the same non-image of C given the same f. And the solution is clearly stored in the *partially-assigned* GGM tree.

2. **Locally Verifiable:** Every step of execution is very simple, making them locally verifiable. In other words, to verify any particular step of the execution, we only need to look at a constant number of assigned values on the *partially-assigned* GGM tree.

Moreover, by choosing $T = 2n \cdot 2^{2n}$, we know an f that is trivially not in the image of $\mathsf{GGM}_T[C]$: the concatenation of all $2n$-length strings.

1.2.3 Finding a Short Description of the History

The downside of choosing $T = 2n \cdot 2^{2n}$ is that the size of the computational history is now exponential in n. The locally verifiable property allows us to use a universal quantifier $(\forall)$ and a $O(\log T)$-bit variable to verify all $O(T)$ steps of the algorithm, but ultimately we need a short $(\text{poly}(n))$ description of the computational history if we want to, for example build a $\mathsf{F\Sigma_2P}$ algorithm.

The authors in [CHR24] appeal to the iterative win-win argument for such a short description: they manage to show that within a large interval of input size, there exists at least one input size n such that the corresponding computational history admits a short description. In fact, the computational history will be the output of a (different) *fully-assigned* GGM tree, leveraging the fact that *fully-assigned* GGM tree has small circuit complexity.

We take a slightly different approach: the key observation is that, by changing traversal order in Korten's algorithm, the resulting computational history (i.e. a *partially-assigned* GGM tree) also has small circuit complexity! More specifically, if we change the traversal order to a post-order traversal (i.e. traverse the left subtree, then the right subtree, and finally the root), the resulting *partially-assigned* GGM tree can be decomposed into $O(n)$ smaller *fully-assigned* GGM trees. See Figure 3 for an illustration. The roots of the *fully-assigned* GGM trees are drawn in circles.

As such, we obtain a short description of the computational history: simply store the roots of all these $O(n)$ *fully-assigned* GGM tree.

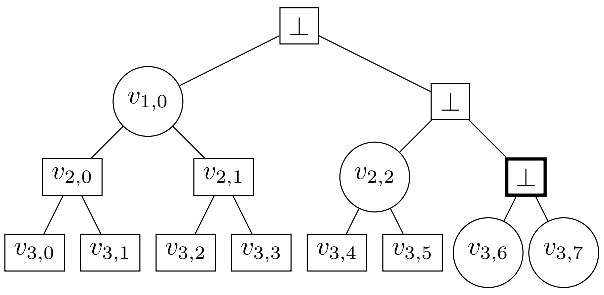

Figure 3: An illustration of the *partially-assigned* GGM tree from running modified Korten's algorithm.

1.2.4 Generalizing to $\mathsf{FS_2P}$

All the ingredients above allow us to build a single-valued $\mathsf{F\Sigma_2P}$ algorithm for Avoid. In order to generalise it to a $\mathsf{FS_2P}$ algorithm, we need a selector algorithm that picks the better witness (in this case, the correct description of the computational history). Now that we have a small description, this turns out to be an easy task. It is now easy to identify a single vertex with different assigned values in the two histories, and traverse down the tree until we hit the leaves, where we know the correct assigned value (i.e. the concatenation of all $2n$-length strings).

2 Preliminaries

Definition 2.1. *Let $s : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in \mathsf{SIZE}[s(n)]$ if L can be computed by circuit families of size $O(s(n))$ for all sufficiently large input size n.*

Definition 2.2. *Let $s : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in i.o.\text{-}\mathsf{SIZE}[s(n)]$ if L can be computed by circuit families of size $O(s(n))$ for infinitely many input size n.*

By definition, we have $\mathsf{SIZE}[s(n)] \subseteq i.o.\text{-}\mathsf{SIZE}[s(n)]$. Hence, circuit lower bounds against $i.o.\text{-}\mathsf{SIZE}[s(n)]$ are stronger and sometimes denoted as *almost-everywhere* circuit lower bound in the literature.

Definition 2.3. *The Range Avoidance (*Avoid*) problem is defined as follows: given as input the description of a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$, for $m > n$, find a $y \in \{0,1\}^m$ such that $\forall x \in \{0,1\}^n : C(x) \neq y$.*

We assume basic familiarity with computational complexity theory, such as complexity classes in the polynomial hierarchy (see e.g. [AB09, Gol08] for references).

Definition 2.4. *Let $T : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in \mathsf{S_2TIME}[T(n)]$, if there exists an $O(T(n))$-time verifier $V(x, \pi_1, \pi_2)$ that takes $x \in \{0,1\}^n$ and $\pi_1, \pi_2 \in \{0,1\}^{T(n)}$ as input, satisfying that:*

- *if $x \in L$, then there exists π_1 such that for every π_2,*
  $V(x, \pi_1, \pi_2) = 1$, and
- *if $x \notin L$, then there exists π_2 such that for every π_1,*
  $V(x, \pi_1, \pi_2) = 0$.

Moreover, we say $L \in \mathsf{S_2E}$ if $L \in \mathsf{S_2TIME}[T(n)]$ for some $T(n) \leq 2^{O(n)}$, and $L \in \mathsf{S_2P}$ if $L \in \mathsf{S_2TIME}[p(n)]$ for some polynomial p.

A search problem Π maps every input $x \in \{0,1\}^*$ into a solution set $\Pi_x \subseteq \{0,1\}^*$. An algorithm A solves the search problem Π on input x if $A(x) \in \Pi_x$.

Definition 2.5 (Single-valued $\mathsf{F\Sigma_2P}$ algorithm)**.** *A single-valued* $\mathsf{F\Sigma_2P}$ *algorithm A is specified by a polynomial $\ell(\cdot)$ together with a polynomial-time algorithm $V_A(x, \pi_1, \pi_2)$. On an input $x \in \{0,1\}^*$, we say that A outputs $y_x \in \{0,1\}^*$, if the following hold:*

1. *There is a $\pi_1 \in \{0,1\}^{\ell(|x|)}$ such that for every $\pi_2 \in \{0,1\}^{\ell(|x|)}$, $V_A(x, \pi_1, \pi_2)$ outputs y_x.*
2. *For every $\pi_1 \in \{0,1\}^{\ell(|x|)}$ there is a $\pi_2 \in \{0,1\}^{\ell(|x|)}$, such that $V_A(x, \pi_1, \pi_2)$ outputs either y_x or $\perp$.*

And we say that A solves a search problem Π if on any input x it outputs a string y_x and $y_x \in \Pi_x$.

Definition 2.6 (Single-valued $\mathsf{FS_2P}$ algorithm)**.** *A single-valued* $\mathsf{FS_2P}$ *algorithm A is specified by a polynomial $\ell(\cdot)$ together with a polynomial-time algorithm $V_A(x, \pi_1, \pi_2)$. On an input $x \in \{0,1\}^*$, we say that A outputs $y_x \in \{0,1\}^*$, if the following hold:*

1. *There is a $\pi_1 \in \{0,1\}^{\ell(|x|)}$ such that for every $\pi_2 \in \{0,1\}^{\ell(|x|)}$, $V_A(x, \pi_1, \pi_2)$ outputs y_x.*
2. *There is a $\pi_2 \in \{0,1\}^{\ell(|x|)}$ such that for every $\pi_1 \in \{0,1\}^{\ell(|x|)}$, $V_A(x, \pi_1, \pi_2)$ outputs y_x.*

And we say that A solves a search problem Π if on any input x it outputs a string y_x and $y_x \in \Pi_x$.

3 Modified Korten's reduction

Notation. We follow the notations from [CHR24] closely: Let s be a n-bit string. We use 0-index where s_0 denotes the first bit of s and s_{n-1} denotes the last bit of s. Let $i < j$, we use $s_{[i,j)}$ to denote the substring of s from the ith bit to the $(j-1)$th bit. We use $s_1 \circ s_2$ to denote the concatenation of two strings s_1 and s_2.

We identify any vertex in a perfect binary tree of height $2n+1$ with a tuple (i,j) where $i \in [0, 2n+1]$ and $j \in [0, 2^i - 1]$, indicating that the vertex is the jth vertex on level i. Note that the two children of (i,j) are $(i+1, 2j)$ and $(i+1, 2j+1)$.

3.1 The GGM Tree

Recall that the GGM tree construction from [GGM86] (vaguely speaking) increases the stretch of a circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$ to arbitrarily long by applying C in a perfect binary tree manner.

Definition 3.1 (The GGM tree construction [GGM86]). *Let $C : \{0,1\}^n \to \{0,1\}^{2n}$ be a circuit. Let $n, T \in \mathbb{N}$ be such that $T \geq 4n$ and let k be the smallest integer such that $2^k n \geq T$. The function $\mathsf{GGM}_T[C] : \{0,1\}^n \to \{0,1\}^T$ is defined as follows.*

Consider a perfect binary tree with 2^k leaves, where the root is on level 0 and the leaves are on level k. Each node is assigned a binary string of length n, and for $0 \leq j < 2^i$, denote $v_{i,j} \in \{0,1\}^n$ the value assigned to the vertex (i,j) (i.e. j-th node on level i). Let $x \in \{0,1\}^n$. We perform the following computation to obtain $\mathsf{GGM}_T[C](x)$: we set $v_{0,0} := x$, and for each $0 \leq i < k, 0 \leq j < 2^i$, we set $v_{i+1,2j} := C(v_{i,j})_{[0,n)}$ (i.e. the first half of $C(v_{i,j})$) and $v_{i+1,2j+1} := C(v_{i,j})_{[n,2n)}$ (i.e. the second half of $C(v_{i,j})$).

Finally, we concatenate all values of the leaves and take the first T bits as the output:

$$\mathsf{GGM}_T[C](x) := (v_{k,0} \circ v_{k,1} \circ \cdots \circ v_{k,2^k-1})_{[0,T)} .$$

For what we need, T is always set to $2n \cdot 2^{2n} = n \cdot 2^{2n+1}$. In other words, the GGM tree will always have height $2n+1$.

It is known that the output of GGM tree has a small circuit [CHR24, Lemma 3.2]. For what we need, we note that the assigned value of any vertex in a GGM tree on a given input has a small circuit.

Lemma 3.2. *Let $\mathsf{GGMEval}(C, T, x, (i,j))$ denote the n-bit assigned value $v_{i,j}$ in the evaluation of the GGM tree $\mathsf{GGM}_T[C](x)$. There is an algorithm running in $\widetilde{O}(|C| \cdot \log T)$ time that, given $C, T, x, (i,j)$, outputs $\mathsf{GGMEval}(C, T, x, (i,j))$.*

Proof sketch. To compute $v_{i,j}$, it suffices to traverse the GGM tree from the root to the vertex (i,j), applying the circuit C in each step. The running time is clearly bounded by the $\widetilde{O}(|C| \cdot \log T)$ since the GGM tree has height $O(\log T)$. □

3.2 Modifying Korten's Reduction

Korten's reduction [Kor21] asserts that given a hard truth table $f \notin \mathrm{Im}(\mathsf{GGM}_T[C])$ and an NP oracle, one can find a non-image for C in $\mathrm{poly}(T, n)$ time.

Note that on a fixed f, Korten's reduction produces the same output. Hence, if we could efficiently simulate the reduction, we obtain an efficient single-valued algorithm. The remaining parts of this section aim to show that Korten's reduction (after our modification) indeed has a small description.

We modify Korten's reduction in the following manner: instead of traversing the perfect binary tree in a simple bottom-up manner, we perform a *post-order traversal* (i.e. traverse the left subtree, then the right subtree and finally the root).

For simplicity, we will fix T to be $n \cdot 2^{2n+1}$ and the perfect binary tree has height $2n+1$. We note that this choice of T (i.e. exponential in n) is the "base case" or "worst case" in the iterative win-win argument in [CHR24]. Since we can handle even the "worst case", we manage to completely bypass the iterative win-win argument.

Fact 3.3. *In a post-order traversal, any root vertex of a subtree is traversed after all other vertices in the subtree. Any vertex in the right subtree is traversed after all vertices in the left subtree.*

For the ease of presentation, we define the total order $<_P$ for all vertices on a perfect binary tree to be the post-order traversal order. In other words, $u_1 <_P u_2$ if and only if u_1 should be traversed before u_2.

Fact 3.4. *Given two vertex $u_1 \neq u_2$ in a perfect binary tree, there is an algorithm that decides whether $u_1 <_P u_2$ or $u_2 <_P u_1$ and runs in time linear in the height of the tree.*

Proof sketch. The algorithm simply finds the lowest common ancestor u_a of u_1 and u_2. By definition of the lowest common ancestor, u_1 and u_2 cannot live in the same proper subtree of u_a.

The vertex living in the left subtree of u_a will be traversed first. If none of them lives in the left subtree of u_a, then one of them must be u_a itself. In this case, the vertex living in the right subtree will be traversed first. □

Algorithm 1: $\mathsf{Korten}'(C, f)$: Modified Korten's reduction

```
Input: $C : \{0,1\}^n \to \{0,1\}^{2n}$ denotes the input circuit, and $f \in \{0,1\}^T \setminus \mathrm{Im}(\mathsf{GGM}_T[C])$ denotes the
       input hard truth table.
Output: A non-output of $C$.
Data: A perfect binary tree of height $2n+1$ that contains the computational history.
1  for $j \leftarrow 0$ to $2^{2n+1} - 1$ do
2      $v_{2n+1,j} \leftarrow f_{[jn,(j+1)n)}$ ;                                // set $f$ to the leaves
3  end
4  for vertex $(i, j)$ in the Post-Order Traversal do
5      Set $v_{i,j}$ be the lexicographically smallest string such that $C(v_{i,j}) = v_{i+1,2j} \circ v_{i+1,2j+1}$ ;  // this
        step requires a NP oracle
6      if $v_{i,j}$ does not exist then
7          Set all remaining vertices $\perp$ ;
8          return $v_{i+1,2j} \circ v_{i+1,2j+1}$ ;
9      end
10 end
11 return $\perp$;
```

3.3 History of $\mathsf{Korten}'(C, f)$

The computational history of $\mathsf{Korten}'(C, f)$ is essentially a *partially-assigned* perfect binary tree of height $2n+1$ where each vertex (i, j) stores a n-bit string $v_{i,j}$ or $\perp$. Unlike [CHR24] where they view the computational history as a long string, we shall keep viewing it as a perfect binary tree and exploit the tree structure. Towards that end, we call it $\mathsf{Histree}(C, f)$.

Definition 3.5 (the computational history of $\mathsf{Korten}'(C, f)$)**.** *Let $n, T \in \mathbb{N}$ be such that $T = n \cdot 2^{2n+1}$. Let $C : \{0,1\}^n \to \{0,1\}^{2n}$ be a circuit, and $f \in \{0,1\}^T$ be a "hard truth table" in the sense that $f \notin \mathrm{Im}(\mathsf{GGM}_T[C])$. The computational history of $\mathsf{Korten}'(C, f)$, denoted as $\mathsf{Histree}(C, f)$, is the partially-assigned perfect binary tree obtained by executing $\mathsf{Korten}'(C, f)$.*

Let $h := \mathsf{Histree}(C, f)$. For any vertex u in the perfect binary tree, we use $h(u)$ to denote the value v_u stored at u. We use $c_L(u)$ and $c_R(u)$ to denote the left child and right child of u.

Definition 3.6 (Proper left children)**.** *Given a set of vertices S from a binary tree, we define the set of proper left children of S to be:*

$$\{u : \exists w \in S, c_L(w) = u, u \notin S\} .$$

The following lemma shows that h has a succinct description.

Lemma 3.7. *Let $n, T \in \mathbb{N}$ be such that $T = 2n \cdot 2^{2n}$. Let $C : \{0,1\}^n \to \{0,1\}^{2n}$ be a circuit, and $f \in \{0,1\}^T$. Let $h := \mathsf{Histree}(C, f)$. h admits a unique description D_h such that:*

- *$|D_h| \le O(n) \cdot \log T$.*
- *There is an algorithm* Eval *that takes in input D_h and any vertex (i, j), outputs $h(i, j)$ in time* $\mathrm{poly}(n) \cdot \log T$.

Proof. We start by describing D_h.

Let $u^* = (i^*, j^*)$ be the vertex where Algorithm 1 finds a solution and terminates. Let $S = \{u_0 = (0,0), u_1, u_2, \ldots, u^*\}$ be the set of vertices on the unique path starting from the root $(0,0)$ to u^*. Note that all vertices in S are assigned $\perp$.

D_h is defined to contain all *proper left children* of S and the right child of u^*, as well as all the stored values in these vertices. We further note that all these vertices have non $\perp$ stored values. In particular, consider any proper left children vertex u_L, it lives in the left subtree of its parent while u^* lives in the right subtree. So u_L must have already been traversed when the algorithm terminates.

Note that D_h contains both children of u^* and hence the output of $\mathsf{Korten}'(C, f)$. It is clear from how we construct D_h that $|D_h| \le O(n) \cdot \log T$ since any path on the tree contains $O(\log T)$ vertices and every vertex stored in D_h carries $O(n)$ bits of information. Also, D_h is uniquely defined for any fixed h.

Next, we show how to efficiently evaluate $h(i, j)$ given any vertex (i, j) in the perfect binary tree. Notice that any subtree in h is also a (smaller) GGM tree. From Lemma 3.2, if we know the stored value of any ancestor of (i, j), we can efficiently (in time $\mathrm{poly}(n) \cdot \log T$) evaluate $h(i, j)$.

Therefore, it suffices to show that for any (i, j), one of its (non $\perp$) ancestors is stored in D_h:

1. If u^* is an ancestor of (i, j), then one of u^*'s children is an ancestor of (i, j) and we know both children are stored in D_h.
2. If (i, j) is an ancestor of u^*, then $h(i, j) = \perp$.
3. Otherwise, let u_a be the lowest common ancestor of u^* and (i, j), and we know that $u_a \in S$. If (i, j) falls in the left subtree of u_a, then the left child of u_a is an ancestor of (i, j) which is stored in D_h. If (i, j) falls in the right subtree of u_a then we argue that $h(i, j) = \perp$. This is because the algorithm stopped at u^* living in the left subtree of u_a, and would not have traversed (i, j).

This concludes the proof. □

The final ingredient we need is that D_h admits a Π_1 verifier on whether its corresponding computational history h is the correct one.

Lemma 3.8 (Π_1 verification of the history)**.** *Let $h := \mathsf{Histree}(C, f)$. There is an oracle algorithm V with input parameter T, n such that the following holds:*

1. *V takes $\tilde{f} \in \{0,1\}^T$ as an oracle, C, $\widetilde{D_h}$ and $w \in \{0,1\}^{2\log T + n}$ as inputs. It runs in* $\mathrm{poly}(n)$ *time.*
2. *D_h defined on $h := \mathsf{Histree}(C, f)$ is the unique string satisfying the following:*

$$V^f(C, D_h, w) = 1, \qquad \forall w \ \in \{0,1\}^{2\log T + n}.$$

Proof. The verifier proceeds in two parts. First part consists of parsing and checking whether $\widetilde{D_h}$ is indeed generated from a valid post-order traversal on the perfect binary tree.

In particular, it reads the terminating vertex u^* based on the two children of u^* stored in $\widetilde{D_h}$, finds the path from $(0,0)$ to u^* and check that all proper left children of vertices on the path are included in $\widetilde{D_h}$ (and of course the right child of u^* should be included). Also stored values of these vertices are non $\perp$.

Upon passing the first part of the verification, we know $\widetilde{D_h}$ corresponds to some *partially-assigned* perfect binary tree $\tilde{h}$ and it remains to check that $\tilde{h}$ is the computational history $\mathsf{Histree}(C, f)$. One should think of the verifier making at most $2^{|w|}$ checks and accepts only if all $2^{|w|}$ check passes.

The verifier V needs to make the following checks. Note that whenever we need some value $v_{i,j}$, we will call $\mathsf{Eval}(\widetilde{D_h}, (i, j))$ for the value. Recall that the total order $<_P$ is defined according to the post-order traversal sequence.

1. The values written on the leaves are indeed f. Hence, for every $j \in [0, 2^{2n+1} - 1]$, check that $v_{2n+1,j}$ is consistent with the corresponding string in f.
2. For every $(i, j) <_P u^*$, $C(v_{i,j}) = v_{i+1,2j} \circ v_{i+1,2j+1}$. (the values are consistent with the children)
3. For every $(i, j) <_P u^*$, for every $x \in \{0, 1\}^n$ that is lexicographically smaller than $v_{i,j}$, $C(x) \neq v_{i+1,2j} \circ v_{i+1,2j+1}$. (the lexicographically first requirement)
4. Let $(i^*, j^*) = u^*$, then for every $x \in \{0, 1\}^n$, $C(x) \neq v_{i^*+1,2j^*} \circ v_{i^*+1,2j^*+1}$. (the two children of u^* form a non-image of C)
5. For every (i, j) where $u^* \leq_P (i, j)$, $v_{i,j} = \perp$.

Each of the above checks is local (requires assigned values of at most 3 vertices) and efficient (runs in time $\mathrm{poly}(n, \log T)$). There are in total $O(T)$ vertices and therefore $O(T)$ tests, which can be implemented with a universal ($\forall$) quantification over at most $2 \log T + n$ bits.

Clearly the correct history h (and therefore its unique description D_h) passes all these checks. Also these checks uniquely determine h as they are essentially enforcing every step of execution of $\mathsf{Korten}'(C, f)$. □

4 Circuit Lower Bound for $\mathsf{S_2E}$

4.1 Single-valued $\mathsf{F\Sigma_2P}$ Algorithm

We start by showing a simple single-valued $\mathsf{F\Sigma_2P}$ Algorithm for Avoid.

Theorem 4.1. *There is a single-valued $\mathsf{F\Sigma_2P}$ algorithm A: when given any circuit $C : \{0, 1\}^n \to \{0, 1\}^{2n}$ as input, $A(C)$ outputs y_C such that $y_C \notin \mathrm{Im}(C)$.*

Proof. On input a circuit $C : \{0, 1\}^n \to \{0, 1\}^{2n}$, let $T = 2n \cdot 2^{2n}$ and $f \in \{0, 1\}^T$ be the concatenation of all $2n$-length bit strings. Let $h = \mathsf{Histree}(C, f)$. $V_A(C, \pi_1, \pi_2)$ is defined as follows: it parses π_1 as D_h and π_2 as w, simulates the verifier $V^f(C, D_h, w)$ in Lemma 3.8. It outputs the non-image of C stored in D_h iff $V^f(C, D_h, w) = 1$. Otherwise it outputs $\perp$.

Note that every position of f can be easily computed since it is just enumerating all $2n$-length strings. Hence the simulation can be done in polynomial time. □

4.2 Single-valued $\mathsf{FS_2P}$ Algorithm

In order to generalise the $\mathsf{F\Sigma_2P}$ algorithm above to a $\mathsf{FS_2P}$ algorithm, we need a 'selector' that chooses the correct D_h when two candidates are given. We formalise such selector in the following lemma.

Lemma 4.2. *Let $n, T \in \mathbb{N}$ be such that $T = 2n \cdot 2^{2n}$. Let $C : \{0, 1\}^n \to \{0, 1\}^{2n}$ be a circuit, and $f \in \{0, 1\}^T$. Let $h := \mathsf{Histree}(C, f)$ and D_h be the succinct description of h defined in Lemma 3.7. Given f as an oracle and two strings π_1, π_2 as additional input, with the promise that $\pi_i = D_h$ for at least one $i \in \{1, 2\}$, there is a deterministic algorithm S such that $S^f(C, \pi_1, \pi_2) = \pi_i$ and runs in time* $\mathrm{poly}(n) \cdot \log T$.

Proof. S starts by parsing π_1, π_2 as D_h. If any of them fail to parse (i.e. the vertices are not derived from a post-order traversal), S simply discards it and output the other one.

Let h_1 and h_2 be the corresponding perfect binary tree generated from π_1 and π_2. Let (i_1^*, j_1^*) be the termination vertex in h_1 and (i_2^*, j_2^*) be the termination vertex in h_2. S will efficiently find a single vertex that contains different stored values in h_1 and h_2. In particular, we consider two cases:

1. $(i_1^*, j_1^*) \neq (i_2^*, j_2^*)$: Without loss of generality, we may assume $(i_1^*, j_1^*) <_P (i_2^*, j_2^*)$. Then we know that $h_2(i_1^*, j_1^*) \neq \bot$.

2. $(i_1^*, j_1^*) = (i_2^*, j_2^*)$: Then they store the same set of vertices in π_1 and π_2, and one of the vertex must have a different stored value.

Now given a vertex (say u) where $h_1(u) \neq h_2(u)$, S proceeds as follows:

1. If u is a leaf vertex, then S checks it against f and decides which is the correct D_h.

2. Otherwise, S checks if $C(h_1(u)) = C(h_2(u))$. If they are indeed pre-images of the same value, S picks π_i for $i \in \{1, 2\}$ such that $h_i(u)$ is the lexicographically smaller one.

   If $C(h_1(u)) \neq C(h_2(u))$ or if $h_2(u) \neq h_1(u) = \bot$, then at least one of u's children (say u') should have a different stored value. S then repeats the whole procedure on u'.

It is clear from the description that S terminates in $O(\log T)$ recursive steps, and the overall running time is $\mathrm{poly}(n) \cdot \log T$. □

Theorem 4.3. *There is a single-valued* $\mathsf{FS_2P}$ *algorithm A: when given any circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$ as input, $A(C)$ outputs y_C such that $y_C \notin \mathrm{Im}(C)$.*

Proof. On input a circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$, let $T = 2n \cdot 2^{2n}$ and $f \in \{0,1\}^T$ be the concatenation of all $2n$-length bit strings. Let $h = \mathsf{Histree}(C, f)$. $V_A(C, \pi_1, \pi_2)$ is defined as follows: it applies the selector algorithm $S^f(C, \pi_1, \pi_2)$ from Lemma 4.2 and obtains $\pi_i = D_h$. It then outputs the non-image of C stored in π_i.

Note that every position of f can be easily computed since it is just enumerating all $2n$-length strings. Hence the simulation can be done in polynomial time. □

4.3 Circuit Lower Bound

Before we get to our circuit lower bound, we need a few results from [Kor21, CHR24]:

Theorem 4.4. *[CHR24, Theorem 2.3] Let $A(x)$ be a single-valued* $\mathsf{FS_2P}$ *algorithm and $B(x, y)$ be an* $\mathsf{FP^{NP}}$ *algorithm, both with fixed output length. The function $f(x) := B(x, A(x))$ also admits an* $\mathsf{FS_2P}$ *algorithm.*

Lemma 4.5. *[Kor21, Lemma 3] Let $n \in \mathbb{N}$. There is a polynomial time algorithm A and an* $\mathsf{FP^{NP}}$ *algorithm B such that the following holds:*

1. *Given a circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$, $A(C)$ outputs a circuit $D : \{0,1\}^n \to \{0,1\}^{2n}$.*

2. *Given any $y \in \{0,1\}^{2n} \setminus \mathrm{Im}(D)$, $B(C, y)$ outputs a string $z \in \{0,1\}^{n+1} \setminus \mathrm{Im}(C)$.*

Definition 4.6. *[CHR24, Section 2.3] For $n, s \in \mathbb{N}$ where $n \leq s \leq 2^n$, the truth table generator circuit $\mathsf{TT}_{n,s} : \{0,1\}^{L_{n,s}} \to \{0,1\}^{2^n}$ maps a stack program of description size $L_{n,s} = (s+1)(7 + \log(n+s))$ into its truth table. Moreover, such circuit can be uniformly constructed in time $\mathrm{poly}(2^n)$.*

The following corollary follows from Theorems 4.3 and 4.4 and Lemma 4.5.

Corollary 4.7. *There is a single-valued* $\mathsf{FS_2P}$ *algorithm A: when given any circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$ as input, $A(C)$ outputs y_C such that $y_C \notin \mathrm{Im}(C)$.*

Corollary 4.8. $\mathsf{S_2E} \not\subset i.o.\text{-}\mathsf{SIZE}[2^n/n]$.

Proof Sketch. Let A be the single-valued algorithm from Corollary 4.7 and set $s := 2^n/n$. Define the language $\mathcal{L}$ such that the truth table of the characteristic function of $\mathcal{L} \cap \{0,1\}^n$ is $A(\mathsf{TT}_{n,s})$. By our choice of s, $L_{n,s} = (s+1)(7+\log(n+s)) < 2^n$ and hence $\mathsf{TT}_{n,s}$ is a valid Avoid instance.

$\mathcal{L} \notin i.o.\text{-}\mathsf{SIZE}[s(n)]$ since any s-size n-input circuit C can be encoded into a stack program of size $L_{n,s}$ bits [FM05].

$\mathcal{L} \in \mathsf{S_2E}$ since one can compute the truth table using algorithm A. □

Remark 4.9. Similar to [CHR24], it is not hard to verify that all our results above relativise.

Acknowledgements

The author was supported by the NUS-NCS Joint Laboratory for Cyber Security, Singapore.

The author would like to thank Eldon Chung, Karthik Gajulapalli, Alexander Golovnev, Sidhant Saraogi and Noah Stephens-Davidowitz for inspiring discussions.

The author would also like to thank Divesh Aggarwal, Lijie Chen, Alexander Golovnev, Hanlin Ren and Sidhant Saraogi for their comments on early drafts of this work.

The author would also like to thank anonymous STOC reviewers for helpful comments.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, USA, 1st edition, 2009. 6

[Cai07] Jin-Yi Cai. $\mathsf{S}_2^p \subseteq \mathsf{ZPP}^{\mathsf{NP}}$. *Journal of Computer and System Sciences*, 73, 02 2007. 2, 3

[CHLR23] Yeyuan Chen, Yizhi Huang, Jiatu Li, and Hanlin Ren. Range avoidance, remote point, and hard partial truth table via satisfying-pairs algorithms. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 1058–1066, New York, NY, USA, 2023. Association for Computing Machinery. 2

[CHR24] Lijie Chen, Shuichi Hirahara, and Hanlin Ren. Symmetric exponential time requires near-maximum circuit size. In *Proceedings of the 56th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2024, to appear. Association for Computing Machinery, 2024. 1, 2, 3, 4, 5, 7, 8, 11, 12

[CT22] Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2022. 2

[CZ19] Eshan Chattopadhyay and David Zuckerman. Explicit two-source extractors and resilient functions. *Annals of Mathematics*, 189(3):653 – 705, 2019. 2

[FM05] Gudmund Skovbjerg Frandsen and Peter Bro Miltersen. Reviewing bounds on the circuit size of the hardest functions. *Information Processing Letters*, 95(2):354–357, 2005. 1, 12

[GG11] Erann Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electron. Colloquium Comput. Complex.*, TR11, 2011. 2

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, aug 1986. 7

[GGNS23] Karthik Gajulapalli, Alexander Golovnev, Satyajeet Nagargoje, and Sidhant Saraogi. Range avoidance for constant depth circuits: Hardness and algorithms. In Nicole Megow and Adam D. Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 65:1–65:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. 2

[GLW22] Venkatesan Guruswami, Xin Lyu, and Xiuhan Wang. Range avoidance for low-depth circuits and connections to pseudorandomness. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 20:1–20:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. 2

[Gol08] Oded Goldreich. Computational complexity: A conceptual perspective. *SIGACT News*, 39(3):35–39, sep 2008. 6

[HNOS96] Lane A. Hemaspaandra, Ashish V. Naik, Mitsunori Ogihara, and Alan L. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM Journal on Computing*, 25(4):697–708, 1996. 2

[ILW23] Rahul Ilango, Jiatu Li, and R. Ryan Williams. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 1076–1089, New York, NY, USA, 2023. Association for Computing Machinery. 2

[IW97] Russell Impagliazzo and Avi Wigderson. P = bpp if e requires exponential circuits: Derandomizing the xor lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 220–229, New York, NY, USA, 1997. Association for Computing Machinery. 1

[Kan82] Ravindran Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1):40–56, 1982. 1

[KKMP21] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou. Total Functions in the Polynomial Hierarchy. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2

[Kor21] Oliver Korten. The hardest explicit construction. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 433–444, 2021. 2, 3, 4, 7, 11

[Li23] Xin Li. Two source extractors for asymptotically optimal entropy, and (many) more. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1271–1281, Los Alamitos, CA, USA, nov 2023. IEEE Computer Society. 2

[MVW99] Peter Bro Miltersen, N. V. Vinodchandran, and Osamu Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In Takano Asano, Hideki Imai, D. T. Lee, Shin-ichi Nakano, and Takeshi Tokuyama, editors, *Computing and Combinatorics*, pages 210–220, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. 1

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. 1

[Rad21] Stanisław P. Radziszowski. Small ramsey numbers. *The Electronic Journal of Combinatorics [electronic only]*, DS01, 2021. 2

[RSW22] Hanlin Ren, Rahul Santhanam, and Zhikun Wang. On the range avoidance problem for circuits. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 640–650, Los Alamitos, CA, USA, nov 2022. IEEE Computer Society. 2

[Sha49] Claude. E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949. 1

[VW23] Nikhil Vyas and Ryan Williams. On Oracles and Algorithmic Methods for Proving Lower Bounds. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, volume 251 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 99:1–99:26, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 3

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il