1 Pseudodeterministic algorithms for $TT^{(s)}$

Recall that we were thinking of proving circuit lower bounds as solving the search problem "find a truth-table with circuit complexity at least $s(\cdot)$", denoted $TT^{(s)}$. Derandomization considerations led us to believe that we should be able to solve this search problem in $\mathcal{FP}^{\mathcal{NP}}$, yielding exponential circuit lower bounds in $\mathcal{E}^{\mathcal{NP}}$.

Extending the previous result? Last time we saw a

pseudodeterministic polytime algorithm hitting dense sets recognizable in $\mathcal{P}$.

This doesn't seem to be enough to find a hard truth-table, since the set of hard truth-tables is in $co\mathcal{NP}$ but doesn't seem to be in $\mathcal{P}$.[1]

A naive hope is that the result will "relativize" in the following sense: if we give both sides an $\mathcal{NP}$ oracle (i.e., the search algorithm and the algorithm recognizing the dense set), the statement will remain true. Since we can recognize hard truth-tables in $\mathcal{P}^{\mathcal{NP}}$, such a relativized result would suffice to yield circuit lower bounds in $\mathcal{BPE}^{\mathcal{NP}}$. Unfortunately such a result isn't currently known.

Open Problem 1. *For any dense set $S \in \mathcal{P}^{\mathcal{NP}}$, construct a pseudodeterministic polytime algorithm with an $\mathcal{NP}$ oracle that solves the search problem of S.*

A relaxation that would suffice is hitting sets recognizable in $co\mathcal{NP}$ (since the set of hard truth-tables is in $co\mathcal{NP}$). We don't know how to do this either in pseudodeterministic polytime with an $\mathcal{NP}$ oracle. So we will solve a more specific problem.

A reminder about pseudodeterminism and single-valuedness. The search problems we'll consider are trivial if we don't insist that the algorithm always outputs the same solution (whp): sample a random string and verify that it belongs to the set of valid solution. And indeed, our deduction of circuit lower bounds does require that the algorithm will have a "canonical" output. To see this, recall that for a given algorithm $A(1^N)$ running in time $\text{poly}(N)$ and producing an N-bit truth-table, we define

$$x \in L_n \iff A(1^N)_x = 1 \,,$$

and without a canonical output the expression "$A(1^N)$" isn't even well-defined.

Last time the notion of a canonical output was formalized as "pseudodeterministic". This time we'll also consider search algorithms in $\mathcal{F}\Sigma_2$, and for such algorithms we formalize the notion as being "single-valued": for every x there is a canonical output v_x such that on every existential guess, the algorithm either prints v_x or rejects.

[1] That is, MCSP isn't known to be in $\mathcal{P}$. Under OWFs it isn't in $\mathcal{P}$, and it's plausible that the problem is $\mathcal{NP}$-complete (or at least close to being so in some sense).

2 Range Avoidance

Instead of solving all explicit construction problems that can be recognized in $co\mathcal{NP}$, we will solve the following specific search problem:

Definition 1 (*Avoid*). *The input is* $D\colon \{0,1\}^n \to \{0,1\}^{2n}$, *and the valid solutions are* $\{0,1\}^{2n} \setminus Range(D) = \{y : \nexists x, D(x) = y\}$.

Note that $Avoid \in co\mathcal{NP}$; hence, for any $\mathcal{P}$-uniform family $\{D_n\}$ of *Avoid* instances, the set of solutions that it defines $S = \left\{y \notin Range(D_{|y|/2})\right\}$ is dense and in $co\mathcal{NP}$. Thus, hitting $\mathcal{P}$-uniform families of *Avoid* instances is indeed a special case of hitting $co\mathcal{NP}$-verifiable dense sets.

As shown by Korten, *Avoid* is already rich enough to capture the problem of finding a hard truth-table. That is, $TT^{(s)}$ reduces to *Avoid*, even for a near-maximal s.

Theorem 2 ([Kor21]). *There is a polynomial-time reduction of* $TT^{(\Theta(2^n/n))}$ *to Avoid.*

Proof. Given $1^{N=2^n}$, consider a circuit D that gets as input a description of a circuit $C\colon \{0,1\}^n \to \{0,1\}$, where the description of C is of size $N/2$, and D outputs the N-bit truth-table of C. Note that $Range(D)$ is the set of all N-bit truth-tables of circuits of size $N/2$, and hence any string outside $Range(D)$ is a truth-table that doesn't have circuits of description size $N/2$, hence also no circuits with $\Theta(2^n/n)$ gates. ■

Note that the reduction above is uniform, and hence solving $\mathcal{P}$-uniform range avoidance instances is already at least as hard as $TT^{(s)}$, for a near-maximal s.

3 The algorithm of CHLR

Chen, Hirahara, Li, and Ren showed that range avoidance can be solved in $\mathcal{FZPP}^{\mathcal{NP}}$, in fact in a smaller class that lies between $\mathcal{FP}^{\mathcal{NP}}$ and $\mathcal{FZPP}^{\mathcal{NP}}$.

Theorem 3 ([CHR23; Li23]). *There is a single-valued* $\mathcal{FS}_2$ *algorithm solving Avoid. Consequently,* $\mathcal{S}_2\mathcal{E} \not\subset \mathcal{SIZE}[\Theta(2^n/n)]$.

We will define the class $\mathcal{S}_2$ later on. For now we should just remember that this class is contained in $\mathcal{FZPP}^{\mathcal{NP}}$, and so it will give us a ppt algorithm with an $\mathcal{NP}$ oracle, and near-maximal circuit lower bounds in $\mathcal{ZPE}^{\mathcal{NP}}$. We will first design an $\mathcal{F}\Sigma_2$ algorithm and then optimize it to $\mathcal{FS}_2 \subseteq \mathcal{FZPP}^{\mathcal{NP}}$.

Note that this result solves *Avoid* regardless of the uniformity of the instance. This overkill has a price: under cryptographic assumptions, the general problem of *Avoid* isn't solvable in $\mathcal{FP}$, and even in $\mathcal{FNP}$ [ILW23; CL24; RWZ25; Ila25]. For $\mathcal{P}$-uniform families, there is evidence for an algorithm in $\mathcal{FP}$ [KS25].

3.1 The first version: An $\mathcal{F}\Sigma_2$ algorithm

We are given $D\colon \{0,1\}^n \to \{0,1\}^{2n}$. For context, recall the naive Σ_2 algorithm: The prover sends a string y, and the verifier checks (using a universal quantifier) that for all x we have $D(x) \neq y$. This algorithm isn't single-valued.

High-level intuition. Think of trying to use D to construct a pseudorandom function, using the construction of [GGM86]. Specifically, as a mental experiment, consider a labeled binary tree $T_{D,\bar{s}}$, whose root is labeled by an initial seed $\bar{s} \in \{0,1\}^n$. Each node labeled with $s \in \{0,1\}^n$ yields a value $D(s) \in \{0,1\}^{2n}$, which is parsed as labels for the two children $(s_1, s_2) \in \{0,1\}^n \times \{0,1\}^n$, and we recurse this for d times, which determines the depth of T_D. As in [GGM86], the PRF $f_{D,\bar{s}}$ takes as input a path in the tree $(b_1, ..., b_d) \in \{0,1\}^d$ and outputs the n-bit string at the bottom layer reached by following the path. Hence, the bottom layer of n-bit strings is the truth-table of $f_{D,\bar{s}}$.

The proof shows a single-valued Σ_2 algorithm *Rec* that, when given input D and oracle access to a candidate f, satisfies the following: If $f \neq f_{D,\bar{s}}$ for all $\bar{s}$, then *Rec* solves *Avoid* for D.[2] The key observation is that if $f \neq f_{D,\bar{s}}$ for all $\bar{s}$, then for some pair of sibling nodes in the tree labeled by (s_1, s_2), there is no label s for the parent such that $D(s) = (s_1, s_2)$, and hence, $(s_1, s_2) \notin \mathit{Range}(D)$. Using the tree structure, the algorithm *Rec* will find the topologically-first nodes where this happens, in Σ_2.

The *Avoid* algorithm invokes *Rec* with a trivial f of exponential size $2^{O(n)}$, such that $f \neq f_{D,\bar{s}}$ for all $\bar{s}$ and query access to f can be simulated easily.

The GGM tree. As a mental experiment, consider the concatenation of all $2n$-bit strings, which is itself a string of length $2n \cdot 2^{2n}$. Partition this string into consecutive substrings of length n, and build a perfect binary tree T whose leaves are labeled by the corresponding n-bit strings (so there are 2^{2n+1} leaves).

The following labeling procedure represents an attempt to present this tree as a GGM tree $T_{D,\bar{s}}$ for D, with some $\bar{s}$ (that is implicitly constructed throughout the labeling procedure). The procedure goes from bottom to top. For each pair of nodes labeled by $y = (y_1, y_2)$, if $y \in \mathit{Range}(D)$ we label the parent by the lex-first pre-image; and if $y \notin \mathit{Range}(D)$, label the parent by $\perp$. Keep doing this, moving up the tree, and whenever one of the siblings is labeled by $\perp$, label the parent by a $\perp$. Observe that:

1. If u is labeled by $\perp$ and its two children are labeled by non-$\perp$, then the two children are a solution. Also, since the pairs of leaves represent all $2n$-bit strings, some u must be labeled by $\perp$.[3]

[2] Thus, this security reduction for the PRF considers stronger adversaries than in [GGM86]; there, the PRF was supposed to be secure against a random f and for polytime adversaries, whereas here we ask if there exists f (given as an oracle) such that a Σ_2 algorithm can distinguish f from $f_{D,\bar{s}}$. (As usual in security reductions, the algorithm *Rec* will try to reconstruct $\bar{s}$ such that $f = f_{D,\bar{s}}$ in a specific way, and if that way doesn't work, then *Rec* solves *Avoid* for D.)

[3] There are many such u's, and at least one in the next-to-bottom level, but we won't use these facts.

2. We can define a notion of "a canonical solution" in the tree, e.g. the topologically-first $\perp$-labeled vertex according to some ordering.[4]

3. Given a purported labeled tree $\tilde{T}$ that is, supposedly, the labeled tree defined above, we can "locally verify" that $T = \tilde{T}$ using D and a universal quantifier. That is, given the labels (s, s_1, s_2) of (u, v, w) in $\tilde{T}$, we can verify that the labeling of u is correct assuming that the labels of v, w are correct: If $s \neq \perp$ we check that $D(s) = (s_1, s_2)$ and that for all $s' < s$ we have $D(s') \neq (s_1, s_2)$; and if $s = \perp$, we check that for all s' it holds that $D(s') \neq (s_1, s_2)$.

A first failed attempt. The prover sends a labeled tree $\tilde{T}$ and the name of the vertex u^*, and the verifier *Rec* uses "forall" quantifiers to check that:

1. The labels of all leaves are the correct ones (using a "forall" quantifier).

2. For every vertex u in $\tilde{T}$, use a "forall s'" quantifier to check that the labeling of u in $\tilde{T}$ is correct assuming that the labels of v, w in $\tilde{T}$ are correct.

3. The label of u^* is $\perp$, and for every $u < u^*$, the label of u in $\tilde{T}$ is not $\perp$.

If the tests pass, the verifier outputs the labels of the children of u^*.

This algorithm is single-valued and outputs $y \notin \mathit{Range}(D)$. To see this, note that only the correct labeled tree T passes the first two tests, in which case the third test ensures that u^* is the topologically first vertex labeled by $\perp$. So *Rec* always outputs the labels of the children of the topologically first solution in T.

The real issue is that the prover sends a huge tree to the verifier, of size $O(n \cdot 2^{2n})$, and hence *Rec* runs in exponential time. The bright side, however, is that the tests performed on $\tilde{T}$ (after the quantified choices are fixed) run in time $\mathrm{poly}(n)$.

Compressing the tree. We redefine the canonical T to have a slightly different labeling, according to the following algorithm. Traverse the binary tree via DFS, label exactly as before, but after you encounter the first $\perp$, label all subsequent nodes by $\perp$.

The key observation, demonstrated in the following proof, is that (1) We erased a lot of information we don't need (i.e., labels of vertices after u^*); (2) The information that remains is compressible – subtrees can be represented by their root.

Fact 4. *Fixing D, there is a description of this T of size $O(n^2)$, and moreover, given D and this description, we can compute the label of any vertex v in time* $\mathrm{poly}(n)$.

Proof. Consider the unique path P from root to u^*, the first $\perp$-labeled vertex. Let S be the set of left children of vertices along P, along with the right child of u^*. Note that

[4] Of course, we can do that without the tree – say, the lex-first solution in $\{0,1\}^{2n}$ – but the tree structure will be handy later.

S-vertices along with all their descendants are labeled by non-$\perp$, and all other vertices in the tree (including ones on P) are labeled by $\perp$.[5]

Observe that given the label of a vertex $v \in S$, we can reconstruct the labels of the entire subtree rooted at v (by running D repeatedly). In other words, the names and labels of vertices in S *uniquely determine* the labels of all vertices in T. Moreover, given the label of a vertex in S and the name of a vertex v in the subtree, we can reconstruct the label of v in time $\text{poly}(n)$ (by traversing the path, running D at each step). □

Here's the modified algorithm. The prover sends a *concise description* of $\tilde{T}$, in the form of vertices in S and their labels, as well as an index u^*, and the verifier *Rec* uses "forall" quantifiers to check that:

1. The pair (S, u^*) uniquely defines a labeled tree $\tilde{T} = \tilde{T}_{S,u^*}$. That is, following the path to u^*, all left children are in S and labeled, and the only other vertex in S is the right child of u^*.

2. The labels of all leaves in $\tilde{T}$ are the correct ones, and the label of every vertex is consistent with the labels of its children. (This is done as before.)

The first test ensures that (S, u^*) define a unique labeled tree $\tilde{T}$, and the second test ensures that $\tilde{T} = T$, using the same analysis as before.[6] Both quantifiers only guess $\text{poly}(n)$ bits, and the final verification runs in time $\text{poly}(n)$.

Recap. The tree T represents an attempt to find $\bar{s}$ such that $f = f_{D,\bar{s}}$, and it is guaranteed to fail (because f contains all $2n$-bit strings). The verifier *Rec* uses quantifiers to, essentially, compute T efficiently and find the topologically-first place where it fails.

3.2 An optimization: An $\mathcal{FS}_2$ algorithm

The definition of $\mathcal{S}_2 \subseteq \Sigma_2 \cap \Pi_2$ is non-obvious. The optimization of *Rec* to work $\mathcal{S}_2$ instead of Σ_2 is interesting not only because it's formally a better result, but also due to the following theorem:

Theorem 5 ([Cai07]). $\mathcal{S}_2 \subseteq \mathcal{ZPP}^{\mathcal{NP}}$.

Recall that derandomization considerations led us to believe we should get exponential circuit lower bounds in $\mathcal{E}^{\mathcal{NP}}$. The tightening to $\mathcal{FS}_2$ gives near-maximal circuit lower bounds in single-valued $\mathcal{FZPP}^{\mathcal{NP}}$; in other words, we're getting a pseudodeterministic algorithm with an $\mathcal{NP}$ oracle, extending [CLO+23].

[5] Any vertex in P is an ancestor of u^*, hence comes after u^* in DFS-order, and thus it is labeled by $\perp$. Also, if $v \notin P$ is a right child of a vertex in P, or a descendant of such a vertex, then v must be labeled by $\perp$ (if $v = u^*$ this is immediate, and otherwise v is a right-child of an ancestor of u^* (or a descendant of a right child), hence comes after u^* in DFS-order). Thus, the only vertices labeled by non-$\perp$ are left children of vertices in P, the right child of u^*, and their descendants.

[6] A slight difference in the analysis is that, in contrast to before, the verifier now does not check that u^* is topologically first. But this is always true for a tree defined by a pair (S, u^*).

Defining $\mathcal{S}_2$. For motivation, consider $L \in \mathcal{NP} \cap co\mathcal{NP}$. We can combine the $\mathcal{NP}$ verifiers for L and for $\bar{L}$ into a single verifier such that for every x,

$$\exists w : V(x,w) = L(x)$$
$$\forall w : V(x,w) \in \{L(x), \perp\}$$

We want to extend this notion to $\Sigma_2 \cap \Pi_2$, but now it's not clear how to combine the verifiers for L and for $\bar{L}$ into a single verifier. To see this, observe that given a witness w for membership and a witness w' for non-membership, the residual decision is either $\forall z,\ V(x,w,z) = 1$ or $\forall z',\ V(x,w',z') = 0$, and it's not clear how to combine these two into a single "$\forall \bar{z},\ V(x,w,\bar{z}) = b$" for some b (with V that is polytime).

The class $\mathcal{S}_2 \subseteq \Sigma_2 \cap \Pi_2$ supports a definition with a single verifier. Intuitively, instead of having one verifier for the "yes" case and another verifier for the "no" case, we now have a single verifier with two provers, *each tasked with the exact same job*.

Definition 6. *We say that $L \in \mathcal{S}_2TIME[T]$ if there is a T-time verifier V such that for every $x \in \{0,1\}^n$:*

1. *There is π_1 such that for every π_2 we have $V(x,\pi_1,\pi_2) = L(x)$.*
2. *There is π_2 such that for every π_1 we have $V(x,\pi_1,\pi_2) = L(x)$.*

As usual $\mathcal{S}_2 = \cup_c \mathcal{S}_2TIME[n^c]$. The definition extends to search problems in the obvious way.

We think of π_1 and π_2 as competing: to convince the verifier, π_1 needs to win against the best π_2. This by itself looks similar to Σ_2, and indeed it's immediate that $\mathcal{S}_2$ has completeness in the Σ_2 sense. The difference is that the soundness requirement is now stricter. Specifically, symmetry mandates that π_2 does the same job as π_1; thus, the same string π_2 that serves as refutation of π_1 also serves as an assertion that $L(x) = b$. That is, there is a refutation that is sent *before seeing* π_1. Indeed, it's instructive to think of the two proofs as being sent in parallel, rather than one after the other. This is the stricter requirement that $\mathcal{S}_2$ poses, compared to $\Sigma_2 \cap \Pi_2$.

Theorem 7. $\mathcal{S}_2 \subseteq \Sigma_2 \cap \Pi_2$.

Proof. To see $\mathcal{S}_2 \subseteq \Sigma_2$, we use the same verifier V. If $x \in L$, then (by the first item) $\exists \pi_1 \forall \pi_2,\ V(x,\pi_1,\pi_2) = 1$. If $x \notin L$ then (by the second item) there is π_2^* such that $\forall \pi_1,\ V(x,\pi_1,\pi_2^*) = 0$; in particular, it does not hold that $\exists \pi_1 \forall \pi_2,\ V(x,\pi_1,\pi_2) = 1$.

To show that $\mathcal{S}_2 \subseteq \Pi_2$ we show that $\neg L \in \Sigma_2$, and since the definition is symmetric (wrt "yes" instances and "no" instances) we can use the same argument. ■

An alternative view of the definition is that V with an input x specifies a matrix with rows indexed by π_1's and columns indexed by π_2's, and each entry indexed by $V(x,\pi_1,\pi_2)$. For every x there is an all-$L(x)$ row and an all-$L(x)$ column. Note that the cases of $x \in L$ and $x \notin L$ are mutually exclusive.[7]

[7] You might encounter a slightly different definition in the literature, in which the first item refers to $x \in L$ and the second refers to $x \notin L$; the two definitions are equivalent, see e.g. [Can96; RS98; Cai07].

Extending the result. Intuitively, an honest prover will send a concise description (S, u^*) of a tree $\tilde{T}$, and the verifier wants to verify $\tilde{T}$ as before. The challenge is that the verifier can't use a "forall" quantifier that comes after seeing $\tilde{T}$: it needs to verify $\tilde{T}$ using a competing proof from the second prover, which *does not depend on the proof of the first prover*. So each prover $b \in \{1,2\}$ is going to send a concise description of a tree $\tilde{T}_b$, and we will cross-check them to decide which $\tilde{T}_b$ is the correct one.

Specifically, recall that to satisfy the $\mathcal{S}_2$ requirements, we may assume wlog that at least one of the trees is the correct one; so catching one of the trees in a lie will suffice. Also note that if the trees disagree, they necessarily disagree on some vertex v in the concise description. We find such v and check whether the two claimed labels $\ell_1 \neq \ell_2$ of v are preimages of the same value, i.e. $D(\ell_1) = D(\ell_2)$; if they are, then the lexicographically-later ℓ_i is the wrong label. Otherwise, the two trees disagree on the labels $\ell'_1 \neq \ell'_2$ of at least one child v' of v; we recurse the check with v',[8] until we reach a vertex whose children are leaves (we can check the labels of leaves directly).

4 Additional facts about *Avoid* and $\mathcal{E}^{\mathcal{NP}}$ lower bounds

Average-case lower bounds. We saw that algorithms for CAPP imply worst-case circuit lower bounds, and I told you that they also imply average-case circuit lower bounds; we will see a proof of this in the final presentations by Heda & Jason. An analogous statement is true for *Avoid*, and it is easier to prove (see [CHL+23]):

Theorem 8. *An algorithm for Avoid implies strong average-case lower bounds, i.e. every circuit of size $2^{\Omega(n)}$ succeeds only on at most $1/2 + 2^{-\Omega(n)}$ of the inputs.*

Proof. As a warm-up let's start with $\delta = .01$. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a δ-uniquely-decodable code, where $\mathsf{Enc}\colon \{0,1\}^N \to \{0,1\}^{\bar{N}}$ and $\bar{N} = O(N)$. Let $\mathsf{tt}\colon \{0,1\}^{N/2} \to \{0,1\}^{\bar{N}}$ be the circuit that takes as input a description of $C\colon \{0,1\}^{\bar{n}} \to \{0,1\}$ and outputs the $\bar{N}$-bit truth-table of C. Define $D\colon \{0,1\}^{N/2} \to \{0,1\}^N$ as

$$D(C) = \mathsf{Dec}(\mathsf{tt}(C)) \,.$$

For any solution $z \notin \mathit{Range}(D)$, its encoding $\mathsf{Enc}(z) \in \{0,1\}^{\bar{N}}$ is a truth-table δ-hard for circuits of size $N/2 = \Omega(\bar{N})$. Otherwise, a circuit C of size $N/2$ computes a δ-approximation of $\mathsf{Enc}(z)$, in which case $\mathsf{Dec}(\mathsf{tt}(C)) = z$, in particular $z \in \mathit{Range}(D)$.

To get $\delta = (1/2 + \epsilon)$ for a small $\epsilon > 0$, we use a $(\delta, \mathrm{poly}(1/\epsilon))$-list-decodable code $(\mathsf{Enc}, \mathsf{Dec})$ with list size $\mathrm{poly}(1/\epsilon)$, and define $D(C, \mathsf{adv}) = \mathsf{Dec}(\mathsf{tt}(C), \mathsf{adv})$, where $\mathsf{adv} \in [\mathrm{poly}(1/\epsilon)]$ is the advice for list-decoding (i.e., an index of a word in the output list of Dec). Let $z \notin \mathit{Range}(D)$. If there's a circuit C of size $N/2$ computing $\mathsf{Enc}(z) \in \{0,1\}^{\bar{N}}$ with success $1/2 + \epsilon$, then for some $\mathsf{adv} \in \{0,1\}^{O(\log(1/\epsilon))}$ it holds that $\mathsf{Dec}(\mathsf{tt}(C), \mathsf{adv}) = z$, in particular $z \in \mathit{Range}(D)$.

Setting $\epsilon = 1/N^{\Omega(1)}$, we can use a code with $\bar{N} = N \cdot \mathrm{poly}(1/\epsilon) = N^{1.01}$ and list-size $\mathrm{poly}(1/\epsilon) < N$ (e.g. [STV01]), in which case the *Avoid* instance maps $N/2 +$

[8] If they disagree on the labels of both children, we pick one arbitrarily and recurse on it.

$O(\log N)$ bits to N bits, and hardness is for $\bar{n}$-bit circuits of size $N/2 = \bar{N}^{1-\Omega(1)} = 2^{\Omega(\bar{n})}$ with average-case succeess bounded by $1/2 + \epsilon = 1/2 + 2^{-\Omega(\bar{n})}$. ■

Improving on brute-force to deduce lower bounds. So far we had to solve *Avoid* in polynomial time (perhaps with an $\mathcal{NP}$ oracle). This seems like a stricter requirement compared to Williams' algorithmic method, which only requires us to show some tiny improvement over a brute-force algorithm (for, say, CAPP).

There is an analogous result for *Avoid*, in which a tiny improvement over a brute-force algorithm (for a somewhat natural computation problem of a circuit-analysis flavor) implies an $\mathcal{FP}^{\mathcal{NP}}$ algorithm for *Avoid*. Moreover, there is a similar framework for proving $\mathcal{E}^{\mathcal{NP}}$ lower bounds, wherein designing such an algorithm (i.e., that slightly improves on brute-force) is sufficient and necessary for the lower bound. We will hear about it in the final presentations, from Rishabh & Jacob.

References

[Cai07] Jin-Yi Cai. "$S_2^p \subseteq ZPP^{NP}$". In: *Journal of Computer and System Sciences* 73.1 (2007), pp. 25–35.

[Can96] Ran Canetti. "More on BPP and the polynomial-time hierarchy". In: *Information Processing Letters* 57.5 (1996), pp. 237–241.

[CHL+23] Yeyuan Chen, Yizhi Huang, Jiatu Li, and Hanlin Ren. "Range Avoidance, Remote Point, and Hard Partial Truth Table via Satisfying-Pairs Algorithms". In: *STOC*. ACM, 2023, pp. 1058–1066. DOI: 10.1145/3564246.3585147.

[CHR23] Lijie Chen, Shuichi Hirahara, and Hanlin Ren. "Symmetric Exponential Time Requires Near-Maximum Circuit Size". In: *CoRR* abs/2309.12912 (2023). DOI: 10.48550/ARXIV.2309.12912. arXiv: 2309.12912. URL: https://doi.org/10.48550/arXiv.2309.12912.

[CL24] Yilei Chen and Jiatu Li. "Hardness of range avoidance and remote point for restricted circuits via cryptography". In: *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*. 2024, pp. 620–629.

[CLO+23] Lijie Chen, Zhenjian Lu, Igor Carboni Oliveira, Hanlin Ren, and Rahul Santhanam. "Polynomial-Time Pseudodeterministic Construction of Primes". In: *arXiv preprint arXiv:2305.15140* (2023).

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to construct random functions". In: *J. ACM* 33.4 (1986), pp. 792–807.

[Ila25] Rahul Ilango. "The Oracle Derandomization Hypothesis is False (And More) Assuming No Natural Proofs". In: *Electronic Colloquium on Computational Complexity: ECCC* (2025).

[ILW23] Rahul Ilango, Jiatu Li, and R. Ryan Williams. "Indistinguishability obfuscation, range avoidance, and bounded arithmetic". In: *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC)*. [2023] ©2023, pp. 1076–1089.

[Kor21] Oliver Korten. "The Hardest Explicit Construction". In: *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2021, pp. 433–444.

[KS25] Oliver Korten and Rahul Santhanam. "How to construct random strings". In: *Proc. 40th Annual IEEE Conference on Computational Complexity (CCC)*. Vol. 339. 2025, Art. No. 35, 32.

[Li23] Zeyong Li. "Symmetric Exponential Time Requires Near-Maximum Circuit Size: Simplified, Truly Uniform". In: *Electron. Colloquium Comput. Complex.* TR23-156 (2023). ECCC: TR23-156. URL: https://eccc.weizmann.ac.il/report/2023/156.

[RS98] Alexander Russell and Ravi Sundaram. "Symmetric Alternation Captures BPP". In: *Comput. Complex.* 7.2 (1998), pp. 152–162.

[RWZ25] Hanlin Ren, Yichuan Wang, and Yan Zhong. "Hardness of Range Avoidance and Proof Complexity Generators from Demi-Bits". In: (2025). arXiv: 2511.14061. URL: https://arxiv.org/pdf/2511.14061.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. "Pseudorandom generators without the XOR lemma". In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 236–266.