Electronic Colloquium on Computational Complexity, Report No. 48 (2022)

On the Range Avoidance Problem for Circuits

Hanlin Ren
University of Oxford
hanlin.ren@cs.ox.ac.uk

Rahul Santhanam
University of Oxford
rahul.santhanam@cs.ox.ac.uk

Zhikun Wang
Xi'an Jiaotong University
nocrizwang@gmail.com

April 4, 2022

Abstract

We consider the *range avoidance* problem (called Avoid): given the description of a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ (where $\ell > n$), find a string $y \in \{0,1\}^\ell$ that is not in the range of C. This problem is complete for the class APEPP that corresponds to explicit constructions of objects whose existence follows from the probabilistic method (Korten, FOCS 2021).

Motivated by applications in explicit constructions and complexity theory, we initiate the study of the range avoidance problem for weak circuit classes, and obtain the following results:

1. Generalising Williams's connections between circuit-analysis algorithms and circuit lower bounds (J. ACM 2014), we present a framework for solving $\mathscr{C}$-Avoid in $\mathsf{FP}^{\mathsf{NP}}$ using circuit-analysis *data structures* for $\mathscr{C}$, for "typical" multi-output circuit classes $\mathscr{C}$. As an application, we present a non-trivial $\mathsf{FP}^{\mathsf{NP}}$ range avoidance algorithm for De Morgan formulas.

   An important technical ingredient is a construction of *rectangular* PCPs of proximity, building on the rectangular PCPs by Bhangale, Harsha, Paradise, and Tal (FOCS 2020).
2. Using the above framework, we show that circuit lower bounds for E^{NP} are equivalent to circuit-analysis algorithms *with* E^{NP} *preprocessing*. This is the first equivalence result regarding circuit lower bounds for E^{NP}. Our equivalences have the additional advantages that they work in both infinitely-often and almost-everywhere settings, and that they also hold for larger (e.g., subexponential) size bounds.
3. Complementing the above results, we show that in some settings, solving $\mathscr{C}$-Avoid would imply breakthrough lower bounds, even for very weak circuit classes $\mathscr{C}$. In particular, an algorithm for AC^0-Avoid with polynomial stretch (i.e., $\ell = \mathrm{poly}(n)$) implies lower bounds against NC^1, and an algorithm for NC^0_4-Avoid with very small stretch (i.e., $\ell = n + n^{o(1)}$) implies lower bounds against NC^1 and branching programs.
4. We show that Avoid is in FNP if and only if there is a propositional proof system that breaks every non-uniform proof complexity generator. This result connects the study of range avoidance with fundamental questions in proof complexity.

ISSN 1433-8092

Contents

1 Introduction

In this paper, we consider the *range avoidance* problem, denoted as Avoid:

Problem 1.1 (Range Avoidance Problem, Avoid)**.** Given the description of a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where $\ell > n$, output any string $y \in \{0,1\}^\ell$ that is not in the range of C. That is, for every $x \in \{0,1\}^n$, $C(x) \neq y$.

The *dual weak pigeonhole principle* [Kra01, Jeř04] states that if N pigeons are placed into M holes where $M \geq 2N$, then there is an empty hole. This principle implies that Avoid is a *total* problem, i.e., it always has a valid solution. But what is the computational complexity of finding this solution?

This problem was studied in [KKMP21] under the name 1-Empty.[1] In their paper, the motivation was to identify natural total search problems in the polynomial hierarchy, in particular $\mathsf{TF\Sigma_2}$. Indeed, it is easy to see that Avoid belongs to (the function version of) $\mathsf{\Sigma_2}$, but it is unknown whether it is in FNP. We may try to solve Avoid by guessing a string $y \in \{0,1\}^\ell$ as an answer, but it seems unclear how to verify that y is not in the range of C without using a universal quantifier. Then, [KKMP21] defines a natural subclass of $\mathsf{TF\Sigma_2}$ called APEPP (Abundant Polynomial Empty Pigeonhole Principle), which is the class of total search problems polynomial-time reducible to Avoid.

The avoidance problem is also motivated by proof complexity and bounded arithmetic, in particular, the proof complexity of the dual weak pigeonhole principle. Jeřábek [Jeř04] defined a theory of bounded arithmetic called APC^1 for formalising probabilistic reasoning by incorporating the dual weak pigeonhole principle as an axiom. Krajíček [Kra01, Kra04] connects the dual weak pigeonhole principle to hard candidates for Extended Frege and stronger proof systems, as well as to the provability of circuit lower bounds.

1.1 Explicit Constructions

Another motivation for studying Avoid, which is more relevant to the current paper, is its connection to *explicit construction* problems. The existence of many combinatorial objects, such as Ramsey graphs, expander graphs, and rigid matrices, are proved by the *probabilistic method* [Erd59, Pin73, Val77]. These probabilistic arguments are able to show that a random object has the desired property with non-zero probability, but are usually unable to explicitly construct such an object. Indeed, many explicit construction problems, such as deterministically constructing Ramsey graphs and rigid matrices, are long-standing open questions.

Arguably, for complexity theorists, the most interesting explicit construction problems are *circuit lower bounds*. It is well-known that almost every Boolean function on n input bits requires circuits of size $\Omega(2^n/n)$ to compute [Sha49], but so far the best lower bound for any explicit function against general circuits is only $5n$ [IM02] or $3.1n$ [FGHK16, LY22], depending on the circuit model.

It was pointed out by Korten [Kor21] that the range avoidance problem nicely captures the complexity of explicit constructions. Most explicit construction problems fall into the category where a "non-random" object of length n can be compressed into strictly less than n bits, and there is an efficient decompression algorithm; see [Kor21, Section 3]. In this case, the problem of constructing a "random" object lies in APEPP, as it suffices to find an object outside the range of the decompression algorithm.

[1] "Empty" stands for "empty pigeonhole principle"; the constant 1 means that the input circuit has stretch at least one bit, i.e., $\ell \geq n + 1$.

Example 1.2. Consider, for example, the problem of proving circuit lower bounds. We want to find a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by circuits of size $s := 2^{n/2}$ (say).

Let $\mathsf{TT} : \{0,1\}^{O(s \log s)} \to \{0,1\}^{2^n}$ be the circuit that takes as input the description of a size-s circuit, and outputs the truth table of this circuit. (Here TT denotes *truth table*.) If we could solve Avoid on the particular instance TT, then we could find a truth table $tt \in \{0,1\}^{2^n}$ without size-s circuits, therefore proving a circuit lower bound. More precisely, solving Avoid for TT in polynomial time is equivalent to proving a circuit lower bound for E, while solving Avoid for TT in $\mathsf{FP}^{\mathsf{NP}}$ is equivalent to proving a circuit lower bound for E^{NP}.

Korten was interested in the *structure* of APEPP. Indeed, one of the main results in [Kor21] was that constructing a hard truth table is complete for APEPP under P^{NP} reductions. In other words, if we could construct a hard truth table in $\mathsf{FP}^{\mathsf{NP}}$, then $\mathsf{APEPP} \subseteq \mathsf{FP}^{\mathsf{NP}}$, which means that any "typical" explicit construction problem can be solved in $\mathsf{FP}^{\mathsf{NP}}$.

We are especially interested in the "easier" regime of APEPP. Let $\mathscr{C}$ be a (multi-output) circuit class, we assume $\mathscr{C}$ is also associated with a stretch function $\ell(n) > n$ such that every circuit in $\mathscr{C}$ maps n bits to $\ell(n)$ bits. Let $\mathscr{C}$-Avoid be the range avoidance problem for $\mathscr{C}$ circuits, we ask:

Question 1.3. *For which circuit classes $\mathscr{C}$ are the $\mathscr{C}$-Avoid problems easy?*

To the best of our knowledge, we are the first to consider the $\mathscr{C}$-Avoid problem for restricted circuit classes $\mathscr{C}$. We think this is an interesting research direction for the following reasons:

- For an explicit construction problem Π, one could identify the weakest circuit class $\mathscr{C}$ such that Π reduces to $\mathscr{C}$-Avoid. Therefore, progress on Question 1.3 implies progress on explicit constructions.
- This consideration reveals some new phenomena. For example, even for very weak circuit classes such as $\mathscr{C} = \mathsf{NC}^0$, solving the $\mathscr{C}$-Avoid problem (with stretch $\ell(n) := n + n^{o(1)}$) implies strong lower bounds that are currently out of reach! (See Theorem 1.17.)

There are many interpretations of the word "easy", but for now, let us think of it as "*provably* in $\mathsf{FP}^{\mathsf{NP}}$". Note that if strong enough circuit lower bounds hold, then by [Kor21], every explicit construction problem is in $\mathsf{FP}^{\mathsf{NP}}$. Therefore we insist that the correctness of the avoidance algorithm *does not rely on unproven assumptions.*

Why $\mathsf{FP}^{\mathsf{NP}}$? There are at least two reasons to study $\mathsf{FP}^{\mathsf{NP}}$ algorithms for Avoid.

- First, $\mathsf{FP}^{\mathsf{NP}}$ is one of the most powerful notions of algorithms that we do not know how to solve Avoid. This is similar to the situation that E^{NP} is one of the biggest complexity classes for which we have no super-polynomial size circuit lower bounds.[2]

  Actually, Avoid can be solved in $\mathsf{FP}^{\mathsf{NP}}$ if and only if E^{NP} cannot be computed by $2^{o(n)}$-size circuits [Kor21]. Therefore, if we could solve $\mathscr{C}$-Avoid in $\mathsf{FP}^{\mathsf{NP}}$ unconditionally for more and more powerful classes $\mathscr{C}$, we could make progress towards the notorious open problem of proving circuit lower bounds against the class E^{NP}.
- Second, APEPP has a very nice structure under $\mathsf{FP}^{\mathsf{NP}}$ reductions. Many reductions among problems in APEPP are only known to be computable in P^{NP}, such as the reductions among Avoid for different stretch functions [KKMP21, Kor21] and the APEPP-completeness of finding

[2] Slightly higher classes, such as $\mathsf{ZPE}^{\mathsf{NP}}$ and $\mathsf{MA\text{-}E}$ (the exponential-time analogue of MA), are known to require super-polynomial size circuits [KW98, BFT98].

a hard truth table [Kor21]. In this paper, we will see more examples where we can make progress when considering $\mathsf{FP}^{\mathsf{NP}}$ algorithms.

For comparison, the structure of APEPP under (say) polynomial-time reductions is less clear. For example, it is not known if finding a hard truth table is APEPP-complete under polynomial-time reductions. It is also open whether Avoid $\in \mathsf{FP}$ or its negation is implied by any "plausible" assumption in complexity theory (or cryptography).

Note that Avoid can be solved in $\mathsf{FZPP}^{\mathsf{NP}}$: simply guess a random string $y \in \{0,1\}^{\ell}$ and use the NP oracle to verify if y is not in the range of the input circuit. Therefore, whether Avoid is in $\mathsf{FP}^{\mathsf{NP}}$ is essentially a derandomisation question.

1.2 The Algorithmic Method

Building on his previous work [Wil13], Williams [Wil14] famously proved that $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$, the first non-uniform lower bound against the notorious circuit class ACC^0. Interestingly, the lower bound is proved by an *algorithmic* method: Williams designed a "non-trivial" satisfiability algorithm for ACC^0 circuits, and then showed that such algorithms imply lower bounds against ACC^0. The only property of ACC^0 that Williams uses is that ACC^0-SAT has a non-trivial algorithm; the algorithm-to-lower-bound connection works for any circuit class satisfying some mild technical conditions.

There have been a long line of subsequent developments of the Algorithmic Method [SW13, BV14, Wil16, Wil18b, COS18, MW20, Wil18a, Che19, CW19, VW20, Vio20, CR20, CLW20, CL21, CLLO21]. A recent highlight is the following result proved in [CLW20]:

Theorem 1.4 ([CLW20], Informal)**.** *There is a language in E^{NP} that does not have sub-exponential size ACC^0 circuits on almost every input length.*

Thinking of circuit lower bounds as explicit construction problems, [CLW20] gave an $\mathsf{FP}^{\mathsf{NP}}$-explicit construction of hard truth tables against sub-exponential size ACC^0 circuits. We can even formulate Theorem 1.4 in the language of circuit range avoidance:

Theorem 1.5 (Theorem 1.4, Reformulated)**.** *Let $s(n) := 2^{n^{o(1)}}$, $\mathsf{TT}_{\mathsf{ACC}^0} : \{0,1\}^{O(s(n)\log s(n))} \to \{0,1\}^{2^n}$ be the circuit that takes as input the description of a size-$s(n)$ ACC^0 circuit, and outputs its truth table. Then, there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for solving* Avoid *on the instance $\mathsf{TT}_{\mathsf{ACC}^0}$.*

Recently, the Algorithmic Method has found applications to another problem: constructing rigid matrices! Alman and Chen [AC19] showed how to construct rigid matrices in $\mathsf{FP}^{\mathsf{NP}}$ with parameters much better than previously known constructions; their results were later improved by [BHPT20, CLW20, HV21, CL21]. The key insight in [AC19] is to treat low-rank matrices as a special type of *circuit class*, and the task of constructing rigid matrices reduces to proving average-case circuit lower bounds against this class.

Given the success of the Algorithmic Method, it is natural to ask the following question:

Question 1.6. *Under which conditions can the Algorithmic Method be used to solve general explicit construction problems, such as* Avoid*?*

1.3 Our Results

In this work, we present new algorithmic and structural results for the range avoidance problem. Along the way, we obtain a version of the Algorithmic Method that *characterises* circuit lower bounds for E^{NP}. We begin by presenting a high-level overview of our results, and then we discuss each result in detail.

An algorithmic method for range avoidance. In our first main result, we present a version of the Algorithmic Method for solving Avoid. Let $\mathscr{C}$ be a multi-output circuit class under some closure properties. We show that if the *Hamming weight estimation* problem for $\mathscr{C}$ has a non-trivial data structure, then the range avoidance problem for $\mathscr{C}$ circuits can be solved in $\mathsf{FP}^{\mathsf{NP}}$. Here, a data structure for Hamming weight estimation for $\mathscr{C}$ has a preprocessing phase and a query phase: During the preprocessing phase, it is given the description of a $\mathscr{C}$ circuit C and produces a data structure DS in $\mathsf{FP}^{\mathsf{NP}}$, i.e., polynomial time with access to an NP oracle. Each query consists of an input x, and we want to estimate deterministically, with the aid of DS, the Hamming weight of $C(x)$ in time faster than brute force.

For comparison, the Algorithmic Method for proving circuit lower bounds works as follows. Let $\mathscr{C}$ be a (single-output) circuit class under some closure properties. If there is a non-trivial CAPP algorithm for $\mathscr{C}$ circuits,[3] then $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathscr{C}$. A CAPP algorithm for $\mathscr{C}$ can be seen as a Hamming weight estimation algorithm (without preprocessing) for $\mathsf{TT}_{\mathscr{C}}$, where $\mathsf{TT}_{\mathscr{C}}$ is the multi-output circuit which takes the description of a $\mathscr{C}$ circuit as input and outputs its truth table. Thus, our result is a generalisation of the Algorithmic Method.

As an application of this result, we show *unconditional* $\mathsf{FP}^{\mathsf{NP}}$ algorithms for the range avoidance problem for De Morgan formulas. In particular, let $s = s(n)$ be a polynomial, C be a function that maps n input bits to $n^{\omega(\sqrt{s}\log s)}$ output bits where each output bit is computed by a De Morgan formula of size $s(n)$, then our algorithm finds a non-output of C in $\mathsf{FP}^{\mathsf{NP}}$.

Circuit lower bounds for E^{NP} are data structures. An easy corollary of our main result is that in the Algorithmic Method, CAPP data structures for $\mathscr{C}$, *even with* E^{NP} *preprocessing*, implies $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathscr{C}$.[4] In our second main result, we show that this is the "complete" Algorithmic Method for proving lower bounds for E^{NP}! For strong enough circuit classes $\mathscr{C}$ (such as TC^0, NC^1, or $\mathsf{P}_{/\mathrm{poly}}$), $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathscr{C}$ if and only if it can be proved by a non-trivial CAPP algorithm with E^{NP} preprocessing.

Of course, it would be nicer to obtain a similar equivalence for weaker circuits classes $\mathscr{C}$, namely those that are unable to compute $\mathsf{MAJORITY}$ and perform hardness amplification [GR08, SV10, GSV18]. We achieve this by considering *strong average-case* circuit lower bounds and CAPP algorithms with inverse-circuit-size error[5]: E^{NP} cannot be $(1/2 + 1/\mathrm{poly}(n))$-approximated by $\mathscr{C}$ if and only if there is a non-trivial CAPP algorithm for $\mathscr{C}$ with inverse-circuit-size error and E^{NP} preprocessing.

Range avoidance for smaller stretch implies breakthrough lower bounds. Note that we ignored the role of the stretch function in the above discussion. Actually, even with "perfect" Hamming weight estimation data structures, our main result only solves the range avoidance problem for circuits that maps n input bits to $n^{1+\epsilon}$ output bits, for every constant $\epsilon > 0$.

How does the stretch function affect the difficulty of (provably) solving Avoid? We show that solving the range avoidance problem for circuits with small enough stretch implies breakthrough lower bounds:

[3] CAPP stands for "circuit acceptance probability problem". A CAPP algorithm for $\mathscr{C}$ is a deterministic algorithm that takes as input a $\mathscr{C}$ circuit C, and outputs an estimation of $\Pr_{x\leftarrow\{0,1\}^n}[C(x) = 1]$ within some additive error.

[4] Here, a CAPP data structure for $\mathscr{C}$ with E^{NP} preprocessing is the following data structure: In the preprocessing phase, we are given the input 1^n, and need to produce a data structure DS in $2^{O(n)}$ time with an NP oracle. In the query phase, we are given a $\mathscr{C}$ circuit C on n input bits, and need to estimate $\Pr_{x\leftarrow\{0,1\}^n}[C(x) = 1]$ in non-trivial time, with the aid of DS. Equivalently, it is a Hamming weight estimation data structure for $\mathsf{TT}_{\mathscr{C}}$.

[5] A CAPP algorithm with inverse-circuit-size error approximates the probability that C accepts a random input within additive error $1/|C|$, where $|C|$ is the size of C.

- Solving AC^0-Avoid with quasi-polynomial stretch (for a small enough quasi-polynomial) implies super-polynomial lower bounds against NC^1.
- Even for the very simple circuit class NC^0_4 (where every output bit only depends on *four* input bits), solving NC^0_4-Avoid with stretch $\ell(n) := n + n^{o(1)}$ implies exponential lower bounds against NC^1 and $\oplus\mathsf{L}_{/\mathrm{poly}}$ (parity branching programs).

We interpret these results as an indication that it may be difficult to generalise Korten's results [Kor21] to restricted circuit classes such as ACC^0 or formulas. Korten showed that Avoid reduces to finding a truth table with large circuit complexity (in P^{NP}); if Formula-Avoid (actually, NC^0_4-Avoid) reduces to finding a truth table with large formula complexity, then formula lower bounds would imply lower bounds for an even stronger model, namely parity branching programs.

Complexity of range avoidance below $\mathsf{FP}^{\mathsf{NP}}$. We also study the complexity of range avoidance w.r.t. algorithms less powerful than $\mathsf{FP}^{\mathsf{NP}}$. As mentioned before, it is unknown whether Avoid $\in \mathsf{FNP}$, Avoid $\in \mathsf{FP}$, or their negations are implied by any plausible assumptions. As far as we know, we do not even have a good idea of what the "ground truth" should be! (For comparison, by Korten's result [Kor21], if one believes circuit lower bounds, one should also believe Avoid $\in \mathsf{FP}^{\mathsf{NP}}$.)

It turns out that the statements "Avoid $\in \mathsf{FNP}$" and "Avoid $\in \mathsf{FP}$" can be characterised by classical notions in complexity theory. In particular, we connect the existence of FNP algorithms for Avoid with the security of *proof complexity generators*, and connect the existence of FP algorithms for Avoid with a version of *time hierarchy theorem* with advice.

We now discuss our results in more detail.

1.3.1 An Algorithmic Method for Range Avoidance

Our first main result is a version of the Algorithmic Method for solving the range avoidance problem. Here, instead of non-trivial circuit-analysis algorithms, we consider *data structures with P^{NP} preprocessing and non-trivial query time.*

For a binary string s, let $\delta(s)$ denote the *relative Hamming weight* of s, i.e., the fraction of bits in s that is equal to 1. For a multi-output circuit class $\mathscr{C}$, let $\mathscr{C}$-HammingHit be the following data structure problem:

(Preprocessing) Given the description of a $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, preprocess C in polynomial time with access to an NP oracle (that is, in P^{NP}), and produce a data structure $\mathsf{DS} \in \{0,1\}^{\mathrm{poly}(\ell)}$.

(Query) Given a string $x \in \{0,1\}^n$, distinguish between the case that $\delta(C(x)) = 1$ and the case that $\delta(C(x)) < 0.01$ in deterministic $\ell/\log^{\omega(1)}\ell$ time with oracle access to DS.

We show that if we want to solve $\mathscr{C}$-Avoid in $\mathsf{FP}^{\mathsf{NP}}$, it suffices to design a data structure for the $\mathscr{C}'$-HammingHit problem, where $\mathscr{C}' := \mathsf{NC}^0 \circ \mathscr{C}$. Here, $\mathscr{C}' = \mathsf{NC}^0 \circ \mathscr{C}$ means that each output gate of a $\mathscr{C}'$ circuit is a function over a constant number of $\mathscr{C}$ circuits.

Theorem 1.7 (Main Result 1, Informal)**.** *Let $\mathscr{C}$ be a (multi-output) circuit class, and $\mathscr{C}' := \mathsf{NC}^0 \circ \mathscr{C}$. Suppose there is a data structure for the $\mathscr{C}'$-HammingHit problem with P^{NP} preprocessing and non-trivial (i.e., $\ell/\log^{\omega(1)}\ell$) query time. Then $\mathscr{C}$-Avoid is in $\mathsf{FP}^{\mathsf{NP}}$.*

Remark 1.8. The power of P^{NP} preprocessing for data structures remains to be investigated. Some examples in the literature where P^{NP} preprocessing seems helpful are:

- The currently fastest data structure for the Online Matrix-Vector Multiplication problem achieves *amortised* $n^2/2^{\Omega(\sqrt{\log n})}$ query time [LW17]. It is implicit in their paper that if we allow P^{NP} preprocessing, then the query algorithm can be made worst-case.
- The optimal expander decomposition can be computed in P^{NP} (see e.g., [PT07]). However, in this case, there are also very good expander decomposition algorithms in deterministic polynomial time [GRST21].

A note on the stretch functions. It is clear that a non-trivial data structure for HammingHit is possible only when $n < \ell/\log^{\omega(1)}\ell$. In the above informal statements, we omitted the stretch of $\mathscr{C}$ and $\mathscr{C}'$ circuits for simplicity. Actually, even assuming the best possible HammingHit data structures, Theorem 1.7 could only solve the range avoidance problem for circuits with stretch $\ell(n) = n^{1+\epsilon}$. We refer to Theorem 3.2 for the precise statement of Theorem 1.7.

Application: Range avoidance for De Morgan formulas. We apply our connection to show a non-trivial $\mathsf{FP}^{\mathsf{NP}}$ algorithm that solves the range avoidance problem for De Morgan formulas.

Theorem 1.9. *Let $s(n)$ be a polynomial, $\mathscr{C}$ be the class of multi-output functions where each output bit is computed by a size-$s(n)$ De Morgan formula, and the number of output bits is at least $\ell := n^{\omega(\sqrt{s}\log s)}$. Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-*Avoid.

Roughly speaking, Theorem 1.9 is proved by the "quantum method" [Tal17] for De Morgan formulas: every function computed by a De Morgan formula of size s has approximate degree $O(\sqrt{s}\log s)$. By Theorem 1.7, it suffices to solve the HammingHit problem for De Morgan formulas. In the preprocessing phase, we compute the low-degree polynomials that approximate each output gate and add them into a single polynomial p of degree $O(\sqrt{s}\log s)$. In the query phase, we use $n^{O(\sqrt{s}\log s)} \ll \ell$ time to evaluate p, which gives a good estimation of the relative Hamming weight.

1.3.2 Equivalences between E^{NP} Circuit Lower Bounds and Non-trivial Derandomisation with Preprocessing

In our second set of results, we show that circuit lower bounds for E^{NP} and CAPP algorithms with E^{NP} preprocessing are equivalent. From Theorem 1.7 we know that a non-trivial GapUNSAT algorithm for $\mathscr{C}$, even *with* E^{NP} *preprocessing*, would imply $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathscr{C}$. We show that for powerful enough circuit classes $\mathscr{C}$ (e.g., TC^0, NC^1, or $\mathsf{P}_{/\mathrm{poly}}$), the converse is also true:

Theorem 1.10 (Main Result 2.1, Informal)**.** *Let $\mathscr{C} \in \{\mathsf{TC}^0, \mathsf{NC}^1, \mathsf{P}_{/\mathrm{poly}}\}$. The following are equivalent:*

- E^{NP} *cannot be computed by polynomial-size $\mathscr{C}$ circuits on almost every input length.*
- *There is a non-trivial* GapUNSAT *algorithm for $\mathscr{C}$ circuits with E^{NP} preprocessing.*

For circuit classes $\mathscr{C}$ that are less powerful (i.e., that might not be able to efficiently compute MAJORITY), we show that *strong average-case* circuit lower bounds against $\mathscr{C}$ and CAPP algorithms for $\mathscr{C}$ with inverse-circuit-size error are equivalent:

Theorem 1.11 (Main Results 2.2, Informal)**.** *Let $\mathscr{C}$ be a "weak" circuit class under some mild closure properties. The following are equivalent:*

- E^{NP} *cannot be $(1/2 + 1/\mathrm{poly}(n))$-approximated by $\mathscr{C}$ circuits on almost every input length.*

- *There is a non-trivial* CAPP *algorithm for $\mathscr{C}$ circuits with* E^{NP} *preprocessing and inverse-circuit-size error.*

Actually, we can show equivalences among a lot of notions, including strong average-case lower bounds for E^{NP}, non-trivial CAPP algorithms with E^{NP} preprocessing, subexponential-time CAPP algorithms with E^{NP} preprocessing, and E^{NP}-computable PRGs. See Theorems 4.17 and 4.19 for details.

It is remarkable that although these equivalences do not refer to AVOID, the most natural way of deriving them seems to go through it. In particular, in the range avoidance problem, it is impossible to estimate the Hamming weight of an ℓ-output circuit in $o(\ell)$ time without preprocessing it, so the preprocessing phase appears naturally. It turns out that adding this preprocessing phase to the (standard) Algorithmic Method makes it an equivalence![6] Our results are also a rare instance where a data structure problem (HammingHit or CAPP with preprocessing) plays a crucial role in a fundamental problem in complexity theory.

One advantage of our equivalence is that it also holds for larger size bounds and the case of infinitely-often lower bounds:

Theorem 1.12 (Informal)**.** *Let $\mathscr{C}$ be a "weak" circuit class under some mild closure properties. The following are equivalent:*

- E^{NP} *cannot be* $(1/2 + 1/2^{n^{o(1)}})$*-approximated by $\mathscr{C}$ circuits of size* $2^{n^{o(1)}}$*.*
- *There is a* CAPP *algorithm for $\mathscr{C}$ circuits of size* $2^{n^{o(1)}}$ *with* $2^{n-n^{\Omega(1)}}$ *query time,* E^{NP} *preprocessing, and inverse-circuit-size error, that works for infinitely many n.*

Remark 1.13 (Equivalences between Derandomisation and Lower Bounds)*.*
Equivalences between derandomisation and lower bounds are known in many settings.

- Impagliazzo, Kabanets, and Wigderson [IKW02] showed that $\mathsf{NEXP} \not\subseteq \mathsf{P}_{/\mathrm{poly}}$ if and only if there is a non-deterministic subexponential-time algorithm for CAPP with $n^{o(1)}$ bits of advice and error $1/6$ that works infinitely often.
- Korten's result [Kor21] can also be interpreted as an equivalence between derandomisation and lower bounds: A full derandomisation of the trivial $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm for AVOID is equivalent to both $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{SIZE}[2^{0.1n}]$ and $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{SIZE}[2^n/3n]$.
- Equivalences between derandomisation and *uniform* lower bounds are also known. Impagliazzo and Wigderson [IW01] showed that $\mathsf{EXP} \neq \mathsf{BPP}$ is equivalent to an infinitely-often, subexponential time derandomisation of BPP on average ($\mathsf{BPP} \subseteq$ i.o.-$\mathsf{heurDTIME}[2^{n^{o(1)}}]$). Williams [Wil16] showed that $\mathsf{NEXP} \neq \mathsf{BPP}$ is equivalent to an infinitely-often, subexponential time nondeterministic derandomisation of BPP on average, with $n^{o(1)}$ bits of advice ($\mathsf{BPP} \subseteq$ i.o.-$\mathsf{heurNTIME}[2^{n^{o(1)}}]_{/n^{o(1)}}$).

In our opinion, compared to the above equivalences, our results have the following features that make them particularly attractive:

- First, they work in both infinitely-often and almost-everywhere settings; in contrast, [IW01] and [Wil16] only hold for infinitely-often lower bounds.
- Second, they scale better with large circuit size bounds (such as $2^{n^{o(1)}}$); no similar equivalences to [IKW02] for $\mathsf{NEXP} \not\subseteq \mathsf{SIZE}[2^{n^{o(1)}}]$ or to [IW01] for $\mathsf{EXP} \not\subseteq \mathsf{BPTIME}[2^{n^{o(1)}}]$ are known.

[6] Note that the *proof* that adding the E^{NP} preprocessing phase still gives us lower bounds is highly non-trivial and requires some new ideas. See Section 1.4 for an overview.

- Third, they are also true for weaker circuit classes such as formulas or ACC^0 circuits; in contrast, the arguments in [Kor21] does not seem to yield any characterisation of the lower bound $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{Formula}[2^{0.1n}]$.
- Finally, our equivalences include both subexponential-time derandomisation and non-trivial derandomisation; none of the equivalences above are known to include non-trivial derandomisation.

An interesting corollary of Theorems 1.10 and 1.11 is the following "speed-up" result for derandomisation with E^{NP} preprocessing:

Corollary 1.14 (Informal)**.** *The following are true:*

- *If there is a non-trivial* GapUNSAT *algorithm for* TC^0 *circuits with* E^{NP} *preprocessing, then there is a subexponential-time* CAPP *algorithm for* TC^0 *circuits with* E^{NP} *preprocessing.*
- *Let* $\mathscr{C}$ *be a "weak" circuit class under some mild closure properties. If there is a non-trivial* CAPP *algorithm for* $\mathscr{C}$ *circuits with* E^{NP} *preprocessing and inverse-circuit-size error, then there is a subexponential-time* CAPP *algorithm for* $\mathscr{C}$ *circuits with* E^{NP} *preprocessing and inverse-circuit-size error.*

Remark 1.15 (Comparison with Other Speed-Ups in Complexity Theory).
Williams [Wil13] showed that if CAPP has a nondeterministic algorithm with non-trivial running time, then CAPP also has a nondeterministic subexponential time algorithm. One caveat of this result is that the speed-up algorithm is only infinitely-often correct, and requires n^ϵ bits of advice. Therefore, the speed-up algorithm does not imply the non-trivial algorithm. In contrast, in Corollary 1.14, the speed-up algorithms always imply the non-trivial algorithms.
Oliveira and Santhanam [OS17] showed a similar speed-up result in learning theory: a typical circuit class is "non-trivially learnable" if and only if it is learnable in sub-exponential time. Their result is proved using the connection between natural proofs and learning [RR97, CIKK16], while our result is a strengthening of the Algorithmic Method.

1.3.3 The Role of the Stretch Function

The input to Avoid is a circuit $C : \{0,1\}^n \to \{0,1\}^{\ell(n)}$. How does the stretch function $\ell(n)$ affect the complexity of Avoid? Clearly, the larger ℓ is, the easier it is to solve Avoid. But is there any *qualitative* difference between (say) $\ell(n) = n + 1$, $\ell(n) = 2n$, and $\ell(n) = n^{100}$?
Korten [Kor21] suggests that the answer might be *no*, at least w.r.t. $\mathsf{FP}^{\mathsf{NP}}$ algorithms. The range avoidance problem for stretch $n + 1$ reduces to the problem of finding a truth table without $2^{0.001n}$-size circuits in P^{NP}; the latter problem trivially reduces to the range avoidance problem for some circuit of stretch n^{100}.
However, when we consider $\mathscr{C}$-Avoid for restricted circuit classes $\mathscr{C}$, we show that the answer is *yes*! We think this is an interesting phenomenon revealed by the investigation of $\mathscr{C}$-Avoid for very weak classes $\mathscr{C}$.

Theorem 1.16 (Informal)**.** *Suppose that for a small enough quasi-polynomial* $\ell(\cdot)$*,* AC^0*-*Avoid *with stretch* ℓ *is solvable in* $\mathsf{FP}^{\mathsf{NP}}$*. Then* $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{NC}^1$*.*

Theorem 1.17 (Informal)**.** *Suppose that* NC^0_4*-*Avoid *with stretch* $\ell := n + n^{o(1)}$ *is solvable in* $\mathsf{FP}^{\mathsf{NP}}$*. Then* E^{NP} *does not have subexponential-size formulas and parity branching programs.*

Roughly speaking, Theorem 1.16 is proved by the subexponential-size simulation of NC^1 circuits by AC^0 circuits [Nep70], and Theorem 1.17 is proved by the randomised encoding techniques of [IK00, IK02, AIK06].

Our results imply that if Korten's result can be generalised to formula complexity (i.e., Formula-Avoid reduces to finding a hard truth table w.r.t. formula complexity in P^{NP}), then formula lower bounds imply parity branching program lower bounds! This raises several interesting open questions: Is there any fundamental reason that Korten's techniques do not work for formula complexity? What is the difference between circuits and formulas that plays a crucial role here? Are these results connected to the open question of constructing randomised encodings for polynomial-size circuits [AIK06]? We leave these questions as interesting research directions.

1.3.4 Is Avoid in FNP or FP?

Finally, we turn to the problem of whether Avoid is in FNP or FP.[7] We note that the results in this section involve some technicalities in the stretch functions (see Section 6 for details), but we ignore this issue in the informal overview. We show that:

Theorem 1.18 (Informal)**.** *The following are true:*

- *There is an* FNP *algorithm for* Avoid *if and only if there is a propositional proof system that breaks every non-uniform proof complexity generator.*
- *There is an* FP *algorithm for (a sparse version of)* Avoid *if and only if a version of time hierarchy for* E *holds with near-maximum advice, i.e.,*

$$\mathsf{E} \not\subseteq \text{i.o.-}\mathsf{DTIME}[2^{n+1}]_{/(2^n-\omega(1))}.$$

Background: proof complexity generators. Let $\ell > n$, a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ is a *proof complexity generator* secure against a propositional proof system, if the proof system cannot efficiently prove *any* string not in the range of C [ABRW04]. More formally, for every $y \in \{0,1\}^\ell$, the (properly encoded version of the) statement "$\forall x \in \{0,1\}^n, C(x) \neq y$" does not have proofs of polynomial length, even though the statement is true for most y.

The study of proof complexity generators is partly motivated by the search for explicit tautologies that are hard for strong propositional proof systems such as Extended Frege [Kra04]. It is also motivated by meta-mathematical questions about circuit lower bounds: are strong circuit lower bounds efficiently provable in propositional proof systems such as Frege and Extended Frege? Razborov has made several conjectures supporting the possibility that the truth table generator[8] is secure against Frege [Raz15], while Krajíček has examined the evidence in favour of the truth table generator being hard even for Extended Frege [Kra04]. In other words, it has been hypothesised that EF cannot efficiently prove super-polynomial circuit lower bounds for *any* truth table.

Krajíček [Kra11] has also studied the possibility that some proof complexity generator is secure against *every* proof system. Conversely, we could ask: Is there a proof system that can break *every* proof complexity generator? This question turns out to be characterised by the statement Avoid $\in$ FNP.

[7] If the following is true, then we say Avoid $\in$ FNP. There is a nondeterministic algorithm that takes a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ as input, where $\ell > n$, such that the following holds. (1) The algorithm accepts at least one nondeterministic branch. (2) On every accepting branch, the algorithm outputs a string $y \in \{0,1\}^\ell$ that is not in the range of C successfully.

[8] The truth table generator is the function TT introduced in Example 1.2. If TT is hard for a propositional proof system, then this proof system cannot efficiently prove circuit lower bounds of *any* truth table.

Discussion: generating K^t-random strings. Along the way, we show that the problem of generating strings with near-maximum time-bounded Kolmogorov complexity (K^t) is complete for (the sparse version of) Avoid under polynomial-time reductions.[9]

Consider the following hypothesis:

Hypothesis 1.19. *There is a deterministic polynomial-time algorithm that given $(1^t, 1^n)$, finds a string $x \in \{0,1\}^n$ such that $\mathrm{K}^t(x)$ is large.*

We show that this hypothesis is equivalent to the aforementioned "time-hierarchy" hypothesis. The plausibility of Hypothesis 1.19 remains to be investigated.

Hypothesis 1.19 is a natural generalisation of circuit lower bounds. If we replace K^t with circuit complexity, we obtain the statement that truth tables with high circuit complexity can be generated in deterministic polynomial time, which is equivalent to circuit lower bounds for E. Is there any formal connection between Hypothesis 1.19 and other hardness assumptions (such as circuit lower bounds or cryptographic assumptions)? We leave this question for future research.

We also define a proof complexity generator based on K^t, and show that the K^t generator is the "hardest" proof complexity generator. A proof system breaks every proof complexity generator if and only if it breaks the K^t generator.

1.3.5 A Rectangular PCP of Proximity

A crucial technical ingredient for proving Theorem 1.7 is a rectangular PCP of proximity (or rectangular PCPP). Here we introduce the rectangular PCPP and discuss its features in the context of the Algorithmic Method. We refer the reader to Section 1.4 for the reason that this notion of PCPP is needed.

PCPs (probabilistically checkable proofs) provide a surprisingly efficient way to verify NP proofs. The PCP theorem [AS98, ALM+98] states that any NP proof can be converted into a polynomially-longer PCP such that a verifier can check its validity by only reading a constant number of bits (at randomly selected locations).

It is often desirable that a PCP has additional properties. One property is shortness: suppose the original NP instance has length n and the NP proof has length m, then the PCP has length $\tilde{O}(n+m)$. Short PCPs are constructed in [BGH+05, BGH+06, Din07]. Another property is proximity [BGH+06]: instead of being in the language, we only verify that the input is *close* to being in the language. The benefit of proximity is that a *super efficient* verifier is now possible: instead of seeing the whole input, the verifier only probes a few bits of both the input string and the proof. We also want the PCP to have projection queries [BV14], which means the circuit that maps the PCP randomness to its query indices is a projection (i.e., has the lowest possible circuit complexity). This is useful in the Algorithmic Method.

The final property we consider is rectangularity, recently introduced in [BHPT20]. In a perfectly rectangular PCP, the proof is a $\sqrt{m} \times \sqrt{m}$ matrix Π, and the PCP randomness seed is partitioned into two parts: $\mathsf{seed.row}$ and $\mathsf{seed.col}$. Each PCP query to the proof matrix is specified by a coordinate (r, c) (where we want to probe $\Pi[r, c]$). Rectangularity means that r only depends on $\mathsf{seed.row}$ and c only depends on $\mathsf{seed.col}$. In other words, the queries of a rectangular PCP is generated as follows:

- First, the verifier reads $\mathsf{seed.row}$ and produces $(r_1, r_2, \ldots, r_q)$ without seeing $\mathsf{seed.col}$.
- Then, the verifier forgets about $\mathsf{seed.row}$, reads $\mathsf{seed.col}$ and produces $(c_1, c_2, \ldots, c_q)$.
- The query locations are $(r_1, c_1), (r_2, c_2), \ldots, (r_q, c_q)$.

[9] Again, we emphasize that this result has some technicalities in the stretch.

We do not know if perfectly rectangular PCPs exist. We can only construct *almost* rectangular PCPs, where the randomness seed also contains a short shared portion $\mathsf{seed.shared}$. The verifier sees both $\mathsf{seed.row}$ and $\mathsf{seed.shared}$ while generating $(r_1, r_2, \ldots, r_q)$, and sees both $\mathsf{seed.col}$ and $\mathsf{seed.shared}$ while generating $(c_1, c_2, \ldots, c_q)$.

The motivation for considering rectangular PCPs in [BHPT20] was to construct rigid matrices (and improve [AC19]). In this paper, we show another application of rectangular PCPs (actually, PCPs of proximity): they are a crucial ingredient in our Algorithmic Method for range avoidance!

In this paper, we construct a short and almost rectangular PCP of proximity with projection queries. Here, the input is also a matrix that is queried in a rectangular fashion; see Definition 2.6 for a precise definition.

Theorem 1.20 (Informal)**.** *For every constant $\tau > 0$ and functions $W_{\mathsf{input}}(n), W_{\mathsf{proof}}(n)$ satisfying some technical conditions, $\mathsf{NTIME}[T(n)]$ has a τ-almost rectangular PCP of proximity with proof length $T(n) \cdot \mathrm{polylog}(T(n))$, input matrix width $W_{\mathsf{input}}(n)$, and proof matrix width $W_{\mathsf{proof}}(n)$. Here, τ-almost rectangularity means that $|\mathsf{seed.shared}| \leq \tau \cdot |\mathsf{seed}|$.*

Moreover, for fixed $\mathsf{seed.shared}$, the maps from $\mathsf{seed.row}$ or $\mathsf{seed.col}$ to the query indices (i.e., r_i or c_i) are projections, computable in polynomial time given $\mathsf{seed.shared}$.

Remark 1.21 (Comparison with [BHPT20])*.* Our rectangular PCP of proximity differs from the rectangular PCP in [BHPT20] in the following ways.

- The biggest difference is that our construction is a PCP of *proximity*. As a result, the input is also treated as a matrix, and its query pattern also has to be rectangular.
- The rectangular PCP in [BHPT20] is *smooth*, i.e., every bit in the proof is queried with equal probability. Smoothness is not required in our application, and our rectangular PCPP has no smoothness guarantee.
- Finally, the input matrix size and the proof matrix size in our rectangular PCPP is flexible, while the proof matrix in [BHPT20] is $\sqrt{m} \times \sqrt{m}$. It is easy to make the proof matrix size flexible, but more care needs to be taken for the input matrix. (See Section 7.1.2 where we artificially define a bijection called bin_{H^m}.) This is quite important as in our application, we need the input matrix width to be as small as possible!

Remark 1.22 (Comparison with [BV14])*.* To reduce the circuit complexity "overhead" of the Algorithmic Method, [BV14] constructed PCPs where the query indices are computable by a projection over seed. To achieve this property, [BV14] needed to use the PCP in [BS08]. Unfortunately, this PCP needs $\mathrm{polylog}(n)$ queries; even worse, this property is broken when we use PCP composition to reduce query complexity to $O(1)$.

However, if we allow the queries to depend arbitrarily on a small portion of seed (namely $\mathsf{seed.shared}$), but has to be a projection over the rest of the bits, then this is also achievable using the PCP in [BGH+06]. The [BGH+06] PCP has the advantage of being almost rectangular. We are also able to compose PCPs now, by simply adding the (very short) randomness of the inner PCP into $\mathsf{seed.shared}$. Thus, the query complexity can be reduced to $O(1)$. It turns out that having such a small portion (i.e., $\mathsf{seed.shared}$) does not hurt the Algorithmic Method at all.

1.4 Technical Overview

In this section, we present an overview of the proof of Theorem 1.7.

It is helpful to review the Algorithmic Method for proving E^{NP} lower bounds. Let $L^{\mathsf{hard}} \in \mathsf{NTIME}[2^n] \setminus \mathsf{NTIME}[o(2^n)]$ be a hard language constructed by the nondeterministic time hierarchy theorem [Zák83]. Let V be the PCP verifier of [BGH+06]; here V is an oracle circuit $V^{(-)} : \{0,1\}^r \to$

$\{0,1\}$. This oracle circuit takes PCP randomness as input (so the input length is $r = n + O(\log n)$), and receives the PCP proof as the oracle.

For a proof oracle $\pi : \{0,1\}^r \to \{0,1\}$, denote $p_{\mathsf{acc}}(\pi) := \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[V^\pi(\mathsf{seed}) \text{ accepts}]$. For every input $x \in \{0,1\}^\star$:

- If $x \in L^{\mathsf{hard}}$, then there is a proof oracle π such that $p_{\mathsf{acc}}(\pi) = 1$.
- If $x \notin L^{\mathsf{hard}}$, then for every proof oracle π, we have $p_{\mathsf{acc}}(\pi) \le 0.01$.

Now, suppose that for every input $x \in L^{\mathsf{hard}}$, there is a proof oracle π such that $p_{\mathsf{acc}}(\pi) = 1$, and in addition, π *can be computed by a* $\mathscr{C}$ *circuit.* (Call this assumption the "easy-witness assumption".) Moreover, suppose that the $\mathsf{GapUNSAT}$ problem for $V^{\mathscr{C}}$ can be solved in $2^r / r^{\omega(1)} < o(2^n)$ time. Then there is a faster nondeterministic algorithm for L^{hard} as follows. Given an input x, we first guess a circuit $\mathscr{C}$ that computes a valid proof oracle π, and use the $\mathsf{GapUNSAT}$ algorithm to distinguish between the case that $p_{\mathsf{acc}}(\pi) = 1$ and that $p_{\mathsf{acc}}(\pi) \le 0.01$.

By the nondeterministic time hierarchy theorem, the above speed-up algorithm has to be incorrect. Therefore, our "easy-witness assumption" has to be false, i.e., there is an input $x \in L^{\mathsf{hard}}$ which does not have valid PCP proofs computable by a small $\mathscr{C}$ circuit.

A naïve attempt. Given a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, our goal is to find a non-output of C in $\mathsf{FP}^{\mathsf{NP}}$. Again, let $L^{\mathsf{hard}} \in \mathsf{NTIME}[\ell] \setminus \mathsf{NTIME}[o(\ell)]$ be the hard language constructed by the nondeterministic time hierarchy.[10] Our "easy-witness" assumption now becomes:

Assumption 1.23. For every $x \in L^{\mathsf{hard}}$, there is a PCP proof for x that is in the range of C.

Now we design a faster nondeterministic algorithm M_{fast} that tries to solve L^{hard}. Let $Q_{\mathsf{PCP}} : \{0,1\}^\ell \to \{0,1\}^{2^{|\mathsf{seed}|}}$ be the circuit such that for every PCP proof $\pi \in \{0,1\}^\ell$ and every seed, the seed-th output of $Q_{\mathsf{PCP}}(\pi)$ is the verifier's output when given π as the PCP proof and seed as the randomness. (Here we interpret seed as both a random string of length $|\mathsf{seed}|$ and an integer in $[2^{|\mathsf{seed}|}]$.) Note that if the PCP is efficient enough, then Q_{PCP} is an NC^0 circuit. Let $C'(x) := Q_{\mathsf{PCP}}(C(x))$. To solve L^{hard}, it suffices to solve the Hamming weight estimation problem for C', i.e., distinguish between $\delta(C'(x)) = 1$ and $\delta(C'(x)) \le 0.01$.

There is a serious problem with this approach: The description length of C is already $\Omega(\ell)$, therefore it is impossible to solve the Hamming weight estimation problem in $o(\ell)$ time.

Idea 1: Make copies. Our first idea is simple but crucial: we pick a large enough number $H = \mathrm{poly}(\ell)$ and make H copies of C. That is, instead of the avoidance problem for C, we consider the avoidance problem for the circuit

$$C^H(x_1, x_2, \ldots, x_H) = (C(x_1), C(x_2), \ldots, C(x_H)).$$

There is a simple $\mathsf{FP}^{\mathsf{NP}}$ reduction from the avoidance problem of C to the avoidance problem of C^H. Suppose $y = (y_1, y_2, \ldots, y_H)$ is not in the range of C^H, then we can use the NP oracle to check whether each y_i is in the range of C, and pick the first y_i that is not. Hence, it suffices to solve the avoidance problem for C^H.

Now, let $L^{\mathsf{hard}} \in \mathsf{NTIME}[H \cdot \ell] \setminus \mathsf{NTIME}[o(H \cdot \ell)]$. Let $Q_{\mathsf{PCP}} : \{0,1\}^{H \cdot \ell} \to \{0,1\}^{2^{|\mathsf{seed}|}}$ be the NC^0 circuit mapping the PCP proof to the verifier's outputs on each seed. It suffices to design a

[10] Note that we have not specified the input length for L^{hard}. We only know that L^{hard} is in non-deterministic ℓ time on this input length. This important issue will be discussed later.

$\mathsf{HammingHit}$ data structure for the circuit $C'(x) := Q_{\mathsf{PCP}}(C^H(x))$. Note that we only need $O(\ell)$ bits to describe C': Q_{PCP} is completely determined by the PCP verifier, and we can use $O(\ell)$ bits to describe C. As $O(\ell) \ll H \cdot \ell$, at least in principle, it could be possible to solve the $\mathsf{HammingHit}$ problem for C' in less than $H \cdot \ell$ time.

But how do we *actually* solve the $\mathsf{HammingHit}$ problem? We need to exploit the structures of the circuit Q_{PCP} (if any)! What property should the PCP have?

Idea 2: Rectangular PCP. Our second idea is to use *rectangular* PCPs. In this overview, let us assume the PCP is *perfectly* rectangular.

We recall the definition of rectangular PCPs. Here, the PCP proof π is an $H \times \ell$ matrix, and our easy-witness assumption becomes that every row of π is in the range of C. The PCP randomness seed is divided into two parts: $\mathsf{seed.row}$ and $\mathsf{seed.col}$. The row index of each query only depends on $\mathsf{seed.row}$, and the column index of each query only depends on $\mathsf{seed.col}$.

We enumerate $\mathsf{seed.row}$. Denote the PCP verifier as V, we want to estimate

$$\Pr_{\mathsf{seed.col}}[V^{\pi}(\mathsf{seed.row}, \mathsf{seed.col}) \text{ accepts}]. \tag{1}$$

As $\mathsf{seed.row}$ is fixed, we now know q rows $r_1, r_2, \ldots, r_q$ such that $V^{\pi}(\mathsf{seed.row}, -)$ will only access these rows of π. Call these rows $\pi_{r_1}, \pi_{r_2}, \ldots, \pi_{r_q} \in \{0,1\}^{\ell}$. Let $Q_{\mathsf{PCP}} : (\{0,1\}^{\ell})^q \to \{0,1\}^{2^{|\mathsf{seed.col}|}}$ be the NC^0 circuit such that for every $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output of $Q_{\mathsf{PCP}}(\pi_{r_1}, \pi_{r_2}, \ldots, \pi_{r_q})$ is 1 if and only if $V^{\pi}(\mathsf{seed.row}, \mathsf{seed.col})$ accepts. Let $x_1, x_2, \ldots, x_q \in \{0,1\}^n$, define the following $\mathsf{NC}^0 \circ \mathscr{C}$ circuit:

$$C'(x_1, x_2, \ldots, x_q) = Q_{\mathsf{PCP}}(C(x_1), C(x_2), \ldots, C(x_q)).$$

We guess the strings $w_1, w_2, \ldots, w_H \in \{0,1\}^n$ such that the i-th row of π is equal to $C(w_i)$. It is easy to see that

$$(1) = \delta(C'(w_{r_1}, w_{r_2}, \ldots, w_{r_q})).$$

Therefore, we can use a $\mathsf{HammingHit}$ data structure for C' to estimate Eq. (1). Note that q is a constant, and $|\mathsf{seed.col}| \approx \log \ell$, so $C' : \{0,1\}^{qn} \to \{0,1\}^{2^{|\mathsf{seed.col}|}}$ is *indeed* small (instead of only having a short description).

To summarise, our speed-up algorithm M_{fast} proceeds as follows. First, we guess the inputs $w_1, w_2, \ldots, w_H$, (implicitly) construct an $H \times \ell$ proof matrix π whose i-th row is equal to $C(w_i)$, and hope that π is a valid PCP proof. Then, we estimate the probability that the PCP verifier accepts. To do so, we enumerate $\mathsf{seed.row}$ and use the $\mathsf{HammingHit}$ data structure to estimate Eq. (1). If for any $\mathsf{seed.row}$ it happens that (1) ≤ 0.01, then we reject; otherwise, we accept.

Since the query algorithm for $\mathsf{HammingHit}$ takes $2^{|\mathsf{seed.col}|}/|\mathsf{seed.col}|^{\omega(1)}$ time, the time complexity of M_{fast} is

$$2^{|\mathsf{seed.row}|} \cdot 2^{|\mathsf{seed.col}|}/|\mathsf{seed.col}|^{\omega(1)} \leq (H\ell)/\log^{\omega(1)} \ell.$$

The "right" time hierarchy theorem. The above avoidance algorithm is only correct on infinitely many input lengths. The reason is that the nondeterministic time hierarchy in [Zák83] only works infinitely often, i.e., for any $\mathsf{NTIME}[o(H\ell)]$ machine M, L^{hard} and M only disagree on infinitely many input lengths.

To obtain an almost-everywhere avoidance algorithm, we follow the ideas of [CLW20]. The crucial observation is that M_{fast} does not guess too many nondeterministic bits. (In the case of the Algorithmic Method, it only guesses a small circuit encoding the PCP proof; in our case, it only guesses $Hn \ll H\ell$ bits.) There is an almost-everywhere nondeterministic time hierarchy against

such machines [FS16]. Let $\mathsf{NTIMEGUESS}[T(N), g(N)]$ denote the class of languages decidable by a nondeterministic machine running in $T(N)$ time and guessing $g(N)$ bits. Then:

Theorem 1.24 ([FS16])**.** *Let $T(N)$ be a time-constructible function such that $N \le T(N) \le 2^{\mathrm{poly}(N)}$. There is a language $L^{\mathsf{hard}} \in \mathsf{NTIME}[T(N)] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}[o(T(N)), N/10]$.*

Since we need to guess Hn bits, we set the input length to be $N := 10Hn$. We also set $T(N)$ to be a slightly super-linear function such that $T(10Hn) \approx H\ell$.

There is a small issue: M_{fast} needs to access the data structure DS for $\mathsf{HammingHit}$. We cannot compute DS inside M_{fast} as it needs an NP oracle, therefore our only option is to hardcode DS as advice for M_{fast}. Fortunately, the above NTIME hierarchy theorem also holds against machines with $N/10$ advice bits:

Theorem 1.25. *Let $T(N)$ be a time-constructible function such that $N \le T(N) \le 2^{\mathrm{poly}(N)}$. There is a language $L^{\mathsf{hard}} \in \mathsf{NTIME}[T(N)] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}[o(T(N)), N/10]_{/(N/10)}$.*

Note that we only need the $\mathsf{HammingHit}$ data structure for the circuit C' whose size is independent of H. By setting H large enough, we can still guarantee that the advice length is $\le N/10 = Hn/10$.[11]

To complete the description of our $\mathsf{FP}^{\mathsf{NP}}$ avoidance algorithm, we still need one ingredient from [CLW20]: a *refuter* for Theorem 1.25. Given 1^N and the code of the machine M_{fast} that attempts to compute L^{hard}, as well as the $N/10$ advice bits, if M_{fast} runs in $o(T(N))$ time and uses at most $N/10$ nondeterministic bits, then the refuter finds an input $x \in \{0,1\}^N$ such that $M_{\mathsf{fast}}(x) \neq L^{\mathsf{hard}}(x)$. The refuter runs in polynomial time with access to an NP oracle.

Our $\mathsf{FP}^{\mathsf{NP}}$ avoidance algorithm is as follows. We first compute the $\mathsf{HammingHit}$ structure DS in $\mathsf{FP}^{\mathsf{NP}}$. We also compute (the code of) the machine M_{fast}. Then we use the refuter to find an input $x_{\mathsf{hard}} \in \{0,1\}^N$ such that $M_{\mathsf{fast}}(x_{\mathsf{hard}}) \neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$. It follows that in any valid proof matrix of $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$, there is some row that is not in the range of C. We can then simply use the NP oracle to pick the first such row.

Rectangular PCP of proximity. There is another issue: Q_{PCP} depends on the input x_{hard}! As x_{hard} depends on DS (recall that x_{hard} is found by the refuter, which takes DS as input), we cannot preprocess $C' = Q_{\mathsf{PCP}} \circ C$ before we know x_{hard}.

Our solution is to use a rectangular PCP of *proximity* (henceforth rectangular PCPP). Recall that a PCPP verifier can only query a small number of bits in both the proof oracle and the input oracle. (As it does not even have time to read the whole input, its query pattern does not depend on it.) In a rectangular PCPP, the input oracle is also accessed in a rectangular fashion. There are three predicates V_{type}, V_{row}, and V_{col}:

- V_{type}, without looking at seed, outputs q symbols, where each symbol is either input or proof.[12]
- V_{row} reads $\mathsf{seed.row}$ and outputs q row indices $r_1, r_2, \ldots, r_q$.
- V_{col} reads $\mathsf{seed.col}$ and outputs q column indices $c_1, c_2, \ldots, c_q$.
- For each query $i \in [q]$, if the i-th symbol is input, then the i-th query asks the (r_i, c_i)-th entry of the input matrix; if the i-th symbol is proof, then the i-th query asks the (r_i, c_i)-th entry of the proof matrix.

[11] In the case of almost rectangular PCPs, we need to hardcode a data structure for every possible value of $\mathsf{seed.shared}$. It is still possible to set the parameters so that the total length of these data structures is $\le N/10$.

[12] Note that we consider perfect rectangularity here. In an almost rectangular PCPP, V_{type} depends on $\mathsf{seed.shared}$, but does not depend on $\mathsf{seed.row}$ and $\mathsf{seed.col}$.

We now revise our speed-up algorithm M_{fast} for L^{hard} using rectangular PCPPs. Given an input $x \in \{0,1\}^N$,[13] we still guess $w_1, w_2, \ldots, w_H$ and construct the PCPP proof matrix π whose i-th row is $C(w_i)$. Also, the input is treated as an $H' \times W'$ matrix[14]; let x_i be the i-th row of the input matrix. Now we estimate the probability that $V^{x,\pi}(\mathsf{seed})$ accepts, where V is the PCPP verifier with oracle access to x and π. After enumerating $\mathsf{seed.row}$, we have fixed q_{proof} rows in the proof matrix and q_{input} rows in the input matrix, where $q_{\mathsf{proof}} + q_{\mathsf{input}} = q$, and the output of $V^{x,\pi}(\mathsf{seed.row}, -)$ only depends on these rows. Let $Q_{\mathsf{PCPP}} : \{0,1\}^{q_{\mathsf{proof}} \cdot n + q_{\mathsf{input}} \cdot W'} \to \{0,1\}^{2^{|\mathsf{seed.col}|}}$ be the circuit that maps these rows to the verifier's outputs on each $\mathsf{seed.col}$,[15] and

$$C'(w_1, w_2, \ldots, w_{q_{\mathsf{proof}}}, x_1, x_2, \ldots, x_{q_{\mathsf{input}}}) = Q_{\mathsf{PCPP}}(C(w_1), C(w_2), \ldots, C(w_{q_{\mathsf{proof}}}), x_1, x_2, \ldots, x_{q_{\mathsf{input}}}).$$

Note that Q_{PCPP} does not depend on the input x.

For each $\mathsf{seed.row}$, we feed the corresponding rows in the proof matrix and the input matrix into C', and use the $\mathsf{HammingHit}$ data structure to estimate the probability over $\mathsf{seed.col}$ that $V^{x,\pi}(\mathsf{seed})$ accepts. The total running time is

$$2^{|\mathsf{seed.row}|} \cdot 2^{|\mathsf{seed.col}|}/|\mathsf{seed.col}|^{\omega(1)} < H\ell/\log^{\omega(1)} \ell.$$

Finally, our $\mathsf{FP}^{\mathsf{NP}}$ avoidance algorithm is the same as before, except that we use the rectangular PCPP in the code of M_{fast}.

2 Preliminaries

We use $\mathcal{U}_n$ to denote the uniform distribution over $\{0,1\}^n$. For a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, denote the range of C as

$$\mathrm{Range}(C) := \{C(x) : x \in \{0,1\}^n\}.$$

The *relative Hamming weight* of a string $x \in \{0,1\}^\ell$, denoted as $\delta(x)$, is the fraction of indices $i \in [\ell]$ such that $x_i = 1$. For two strings $x, y \in \{0,1\}^\ell$ of equal length, the *relative Hamming distance* between x and y, denoted as $\delta(x,y)$, is the fraction of indices $i \in [\ell]$ for which $x_i \neq y_i$.

We say a function $f : \mathbb{N} \to \mathbb{N}$ is *good* if there is a Turing machine that, given the input n (in binary), outputs the value $f(n)$ (also in binary), and runs in time at most $\mathrm{poly}(\log n, \log f(n))$. (This is just a somewhat arbitrary definition of "functions that are not too pathological".)

2.1 The Computational Models

In this paper, we need to deal with two (nondeterministic) computational models: standard multi-tape Turing machines (TMs) and Random Access Turing Machines (RTMs). The difference between TMs and RTMs is the following: for each work tape of an RTM, there is a corresponding address tape such that the head of the work tape is always in the cell whose index is the contents of the address tape [GS89]. We need to be careful about which machine model we are using. For example, the query algorithms for the data structures run in the RTM model, while the highly-efficient PCPP [BGH+05] uses the TM model.

Let $T(n)$ be a good function, we use $\mathsf{NTIME}_{\mathrm{TM}}[T(n)]$ and $\mathsf{NTIME}_{\mathrm{RTM}}[T(n)]$ to denote the set of languages computable in nondeterministic $T(n)$ time on a multi-tape Turing machine and an RTM respectively. We need the following result to simulate one machine model by the other efficiently.

[13] Note that a PCPP could only distinguish between $x \in L$ and x being *far* from L. Thus, we need to apply an error-correcting code to the input. For simplicity, we still use x to denote the encoded input.

[14] A technicality here is that we want to set W' to be as small as possible, as the size of C' is proportional to W'. It turns out that we can achieve $W' = n \cdot \mathrm{polylog}(\ell)$.

[15] Actually, Q_{PCPP} also depends on $O(q)$ *parity-check* bits. We ignore this technical detail in the overview.

Theorem 2.1 ([GS89]). $\bigcup_{c\geq 1} \mathsf{NTIME}_{\mathrm{TM}}[n\log^c n] = \bigcup_{c\geq 1} \mathsf{NTIME}_{\mathrm{RTM}}[n\log^c n]$.

By a padding argument, for some absolute constant $c \geq 1$, for every good function $T(n)$,

$$\mathsf{NTIME}_{\mathrm{TM}}[T(n)] \subseteq \mathsf{NTIME}_{\mathrm{RTM}}[T(n)\log^c T(n)],$$
$$\text{and } \mathsf{NTIME}_{\mathrm{RTM}}[T(n)] \subseteq \mathsf{NTIME}_{\mathrm{TM}}[T(n)\log^c T(n)].$$

2.2 Machines That Take Advice

Let C be a complexity class and $f : \mathbb{N} \to \mathbb{N}$. Denote $\mathsf{C}_{/f}$ the class of functions computable by a C machine with $f(n)$ bits of advice [KL80]. More formally, a language L is in $\mathsf{C}_{/f}$ if there is another language $L' \in \mathsf{C}$ and a sequence of "advice" strings $\{\alpha_n \in \{0,1\}^{f(n)}\}_{n\in\mathbb{N}}$ such that

$$\forall n \in \mathbb{N}, x \in \{0,1\}^n, x \in L \iff (1^n, x, \alpha_n) \in L'.$$

Two instances of the above notation in this paper are:

- $\mathsf{NTIMEGUESS}[T(n), n/10]_{/(n/10)}$ (Theorem 2.3): languages computed by a nondeterministic machine in $T(n)$ time, using $n/10$ nondeterministic bits and $n/10$ advice bits.
- $\mathsf{DTIME}[2^{n+1}]_{/(2^n-cn)}$ (Theorem 6.10): languages computed in deterministic 2^{n+1} time with *near-maximum* $(2^n - cn)$ advice bits.

2.3 Error-Correcting Codes

We need the following standard construction of error-correcting codes.

Theorem 2.2 ([Spi96]). *There is a* $\mathrm{GF}(2)$*-linear error-correcting code* $(\mathsf{Enc}, \mathsf{Dec})$ *with constant rate and constant relative distance. Moreover, both* Enc *and* Dec *are computable in linear time.*

2.4 An Almost-Everywhere NTIME Hierarchy with a Refuter

We need the almost-everywhere NTIME hierarchy against bounded nondeterminism [FS16], which has an $\mathsf{FP}^{\mathsf{NP}}$ refuter as shown in [CLW20]. Let $T(n), G(n)$ be good functions, we define $\mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T(n), G(n)]$ to be the class of languages accepted by a nondeterministic RTM in $T(n)$ time with $G(n)$ nondeterministic bits.

Theorem 2.3 ([FS16, CLW20]). *Let c be a large universal constant, $T : \mathbb{N} \to \mathbb{N}$ be a good function such that $n\log^{c+1} n \leq T(n) \leq 2^{\mathrm{poly}(n)}$. There is a language*

$$L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathrm{TM}}[T(n)] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T(n)/\log^c T(n), n/10]_{/(n/10)}.$$

Moreover, there is an algorithm $\mathcal{R}$ (the "refuter") such that the following holds.

(Input) *$\mathcal{R}$ receives three inputs $(1^n, M, \alpha)$, where M is a nondeterministic RTM and $\alpha \in \{0,1\}^{n/10}$ is an advice string. It is guaranteed that M runs in $T(n)/\log^c T(n)$ time and uses at most $n/10$ nondeterministic bits; moreover, the description length of M is $O(1)$.*

(Output) *For every fixed M, every sufficiently large n, and every advice $\alpha \in \{0,1\}^{n/10}$, $\mathcal{R}(1^n, M, \alpha)$ outputs a string $x \in \{0,1\}^n$ such that $M(x;\alpha) \neq L^{\mathsf{hard}}(x)$.*

(Complexity) *$\mathcal{R}$ runs in $\mathrm{poly}(T(n))$ time with adaptive access to an NP oracle.*

For completeness, we include a proof of Theorem 2.3 in Appendix A.1, showing that the time hierarchy and the refuter could also deal with the $N/10$ advice bits.

2.5 Probabilistically Checkable Proof of Proximity

In this sub-section, we introduce the variants of PCPs that we will use in this paper: PCPs of proximity (PCPPs), rectangular PCPPs, and PCPPs with randomness-oblivious predicates.

We begin by recalling the formalism of efficient PCPP verifiers. Note that PCPP verifiers are usually defined for pair languages where each input is split into two parts $(x^{\mathsf{exp}}, x^{\mathsf{imp}})$ (see, e.g., [BGH+06]). In that setting, the verifier is given x^{exp} explicitly and x^{imp} implicitly (i.e., as an oracle). When we consider pure languages (i.e., languages where x^{exp} is always the empty string), we will denote x^{imp} as simply x.

Definition 2.4 (Efficient PCP of Proximity Verifiers)**.** Let $r = r(n)$ and $q = q(n)$ be good functions. A PCPP verifier VPCPP for a language L with *randomness complexity* r and *query complexity* q is specified by a tuple of machines $(V_{\mathsf{type}}, V_{\mathsf{index}}, V_{\mathsf{dec}})$. Given oracle access to an input x and a proof π, it verifies whether $x \in L$ as follows.

- VPCPP draws a random string $\mathsf{seed} \in \{0,1\}^r$ and generates
  - $(\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) \leftarrow V_{\mathsf{type}}(\mathsf{seed})$, and
  - $(i[1], i[2], \ldots, i[q]) \leftarrow V_{\mathsf{index}}(\mathsf{seed})$.

  Here, for each $j \in [q]$, $\mathsf{itype}[j] \in \{\mathsf{input}, \mathsf{proof}\}$ indicates whether the j-th query is on the input oracle x or the proof oracle π. If $\mathsf{itype}[j] = \mathsf{input}$, the answer of the j-th query is $\mathsf{ans}_j := x_{i[j]}$; if $\mathsf{itype}[j] = \mathsf{proof}$, the answer of the j-th query is $\mathsf{ans}_j := \pi_{i[j]}$.
- Finally, $V_{\mathsf{dec}}(\mathsf{seed}, \mathsf{ans}_1, \ldots, \mathsf{ans}_q)$ decides whether to accept Π or not.

We say V_{dec} is the *decision predicate* of VPCPP, and the size of V_{dec} (as a circuit) is the *decision complexity* of VPCPP.[16]

We denote the oracle $x \circ \pi$ as Π. Sometimes we also denote x as Π_{input} and π as Π_{proof}, to emphasize that they are *oracles* (rather than strings). We say $\mathsf{VPCPP}^{\Pi}(\mathsf{seed}) = 1$ if $V_{\mathsf{dec}}(\mathsf{seed}, \mathsf{ans}_1, \ldots, \mathsf{ans}_q)$ accepts, and $\mathsf{VPCPP}^{\Pi}(\mathsf{seed}) = 0$ otherwise. (Note that the behaviour of the verifier is completely deterministic once seed is determined.)

Definition 2.5 (PCP of Proximity for Pure Languages)**.** For functions $s, \delta : \mathbb{N} \to [0,1]$, a verifier VPCPP is a PCP of proximity for a pure language L with *proximity parameter* δ and *soundness error* s if the following two condition holds:

Completeness: If $x \in L$, then there is an oracle π such that

$$\Pr_{\mathsf{seed} \leftarrow \{0,1\}^{r(n)}}[\mathsf{VPCPP}^{x \circ \pi}(\mathsf{seed}) \text{ accepts}] = 1.$$

Soundness: If x is δ-far from being in L (that is, for every $x' \in L \cap \{0,1\}^n$, the relative Hamming distance between x and x' is at least δ), then for every oracle π,

$$\Pr_{\mathsf{seed} \leftarrow \{0,1\}^{r(n)}}[\mathsf{VPCPP}^{x \circ \pi}(\mathsf{seed}) \text{ accepts}] < s(n).$$

[16] In literature, it is more common to define V_{dec} as a circuit that only takes $\mathsf{ans}_1, \ldots, \mathsf{ans}_q$ (but not seed) as inputs. The circuit V_{dec} is outputted by the PCPP verifier *after seeing* seed. However, we find our formulation (where V_{dec} takes seed as input) more convenient to use.

We will also consider a stronger requirement called *robust soundness*. Roughly speaking, it requires that the bits that the decision predicate reads are *far from* being accepted (instead of only being rejected), i.e., we need to flip at least a ρ fraction of bits read by the decision predicate in order to make it accept. Formally:

Robust soundness: If y is δ-far from being in L, then for every oracle π,

$$\Pr_{\mathsf{seed}\leftarrow\{0,1\}^{r(n)}}[\exists a \in \{0,1\}^q, \text{ s.t. } V_{\mathsf{dec}}(\mathsf{seed}, a) = 1 \text{ and the relative Hamming distance between } \mathsf{ans} \text{ and } a \text{ is } \leq \rho] \leq s.$$

Here, ρ is the *robustness parameter* of VPCPP.

Now we define *rectangular PCPPs*, a crucial technical notion in our proof. Roughly speaking, a rectangular PCPP treats both its input oracle Π_{input} and its proof oracle Π_{proof} as matrices and makes queries in a "rectangular" fashion.

For a perfectly rectangular PCPP, its randomness is divided into a *row* part and a *column* part, where the row indices of the queries only depend on the row part, and the column indices of the queries only depend on the column part. However, it is unclear whether perfectly rectangular PCPPs exist. Therefore, we consider a relaxed notion of *almost* rectangular PCPP, where the randomness also contains a short *shared* part that can influence both row indices and column indices.

Definition 2.6 (Rectangular PCP of Proximity)**.** Let $H_{\mathsf{input}}, W_{\mathsf{input}}, H_{\mathsf{proof}}, W_{\mathsf{proof}} : \mathbb{N} \to \mathbb{N}$ be good functions such that $H_{\mathsf{input}} \cdot W_{\mathsf{input}} = O(n)$, and let $\tau > 0$ be a constant. A verifier VPCPP is τ-*almost rectangular* with an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ input oracle, an $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ proof oracle, row randomness $r_{\mathsf{row}}(n)$, column randomness $r_{\mathsf{col}}(n)$, and shared randomness $r_{\mathsf{shared}}(n)$, if the following holds.

(Randomness) The verifier randomness seed has length $r(n) := r_{\mathsf{row}}(n) + r_{\mathsf{col}}(n) + r_{\mathsf{shared}}(n)$. It consists of three parts

$$\mathsf{seed.row} \in \{0,1\}^{r_{\mathsf{row}}(n)}, \mathsf{seed.col} \in \{0,1\}^{r_{\mathsf{col}}(n)}, \text{ and } \mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}(n)}.$$

Moreover, $r_{\mathsf{shared}}(n) \leq \tau \cdot r(n)$.

(Query generation) There are three functions V_{type}, V_{row} and V_{col} such that the query locations of VPCPP are generated as follows:

- Let $(\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) \leftarrow V_{\mathsf{type}}(\mathsf{seed.shared})$ where $\mathsf{itype}[i] \in \{\mathsf{input}, \mathsf{proof}\}$ for all $i \in [q]$.
- Let $(\mathsf{irow}[1], \mathsf{irow}[2], \ldots, \mathsf{irow}[q]) \leftarrow V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared})$.
- Let $(\mathsf{icol}[1], \mathsf{icol}[2], \ldots, \mathsf{icol}[q]) \leftarrow V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared})$.
- For every $j \in [q]$:
  - If $\mathsf{itype}[j] = \mathsf{proof}$, then the j-th query probes the $i[j]$-th bit of Π_{proof}, where
  $$i[j] := \mathsf{irow}[j] \cdot W_{\mathsf{proof}} + \mathsf{icol}[j].$$
  - If $\mathsf{itype}[j] = \mathsf{input}$, then the j-th query probes the $i[j]$-th bit of Π_{input}, where
  $$i[j] := \mathsf{irow}[j] \cdot W_{\mathsf{input}} + \mathsf{icol}[j].$$

∗ In the case that $\mathsf{itype}[j] = \mathsf{input}$, it might be possible that $\mathsf{irow}[j] = \perp$ or $\mathsf{icol}[j] = \perp$ or $i[j] \notin [0, n)$. In this case, we assume the j-th query answer is $\perp$.

Ideally, the decision predicate of a rectangular PCPP should only depend on seed.shared, i.e., it should be a "randomness-oblivious predicate" (ROP, [BHPT20]). However, we (as well as [BHPT20]) do not know how to build rectangular PCPPs whose decision predicate does not depend on seed.row and seed.col at all. It turns out that the following relaxation is achievable: besides seed.shared, the decision predicate is also allowed to take as input some parity-check bits, where each parity-check bit is the XOR of a subset of bits in seed.row and seed.col, and this subset can be computed efficiently from seed.shared.[17]

Definition 2.7 (PCPP with Randomness-Oblivious Predicates)**.** Let VPCPP be a rectangular PCPP with decision predicate V_{dec}. If there are p parity-check functions $pc_1, pc_2, \ldots, pc_p$ over seed, such that each pc_i is the XOR of some subset of bits in seed, and Dec takes $\mathsf{ans} := (\mathsf{ans}_1, \mathsf{ans}_2, \ldots, \mathsf{ans}_q)$ and $P := (pc_1(\mathsf{seed}), pc_2(\mathsf{seed}), \ldots, pc_p(\mathsf{seed}))$ as inputs, then we say VPCPP has a *randomness-oblivious predicate* (ROP) with parity-check complexity p. (In other words, if we are given $pc_1(\mathsf{seed})$, $pc_2(\mathsf{seed})$, …, $pc_p(\mathsf{seed})$, then V_{dec} does not depend on seed.row or seed.col any more.)

Without loss of generality, we may assume that a decision circuit VDec is outputted after VPCPP sees seed.shared, and each pc_i is a parity function over seed.row and seed.col.

Remark 2.8. For clarity, we will now present the streamlined procedure of how a rectangular PCPP with ROP works:

1. Sample shared randomness $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$. Based on it,
   (a) Construct a decision predicate circuit $\mathsf{VDec} := V_{\mathsf{dec}}(\mathsf{seed.shared})$.
   (b) Construct the parity-checks functions
   $$(pc_1, \ldots, pc_p) := (pc_1(\mathsf{seed.shared}), \ldots, pc_p(\mathsf{seed.shared})).$$
   Here, each pc_i is a PARITY function over (a subset of indices in) seed.row and seed.col.
   (c) Compute proof types
   $$(\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) := V_{\mathsf{type}}(\mathsf{seed.shared}).$$
2. Sample row randomness $\mathsf{seed.row} \in \{0,1\}^{r_{\mathsf{row}}}$. Construct proof row locations
   $$(\mathsf{irow}[1], \mathsf{irow}[2], \ldots, \mathsf{irow}[q]) := V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared}).$$
3. Sample column randomness $\mathsf{seed.col} \in \{0,1\}^{r_{\mathsf{col}}}$. Construct proof column locations
   $$(\mathsf{icol}[1], \mathsf{icol}[2], \ldots, \mathsf{icol}[q]) := V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared}).$$
4. Compute randomness parity checks $PC := (pc_1(\mathsf{seed}), pc_2(\mathsf{seed}), \ldots, pc_p(\mathsf{seed}))$.
5. Compute $\mathsf{ans} := (\mathsf{ans}_1, \mathsf{ans}_2, \ldots, \mathsf{ans}_q)$ as in Definition 2.6.
6. Output the result of the computation $\mathsf{VDec}(\mathsf{ans}, PC)$.

We will rely on the following rectangular PCP of proximity:

[17] In general, we could define ROP for *arbitrary* PCPPs: The seed is partitioned into a short *aware* part and the rest *oblivious* part, and V_{dec} only depends on the aware part and a few parity-check bits over the oblivious part. In this paper, we only consider ROP for *rectangular* PCPPs where the aware parts (for ROP) and the shared parts (for rectangularity) are the same.

Soundness error	s
Proximity parameter	δ
Total randomness	$\log T(n) + O(\tau^{-1} \log\log T(n))$
Shared randomness	$(\tau/2)\log T(n) + O(\log\log T(n))$
Row randomness	$h_{\mathsf{proof}} - (\tau/4)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (\tau/4)\log T(n)$
Query complexity	$O(1)$
Parity check complexity	$O(1)$
Decision complexity	$\mathrm{polylog}(T(n))$

Table 1: Parameters of the rectangular PCPP in Theorem 2.9.

Theorem 2.9 (See Theorem 7.11 and Corollary 7.12)**.** *The following holds for every constants $\tau, s, \delta > 0$. Let $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$, be good functions such that $n \le T(n) \le 2^{\mathrm{poly}(n)}$, $\tau \log T(n) \le w_{\mathsf{proof}}(n) \le \log T(n)$, and $w_{\mathsf{input}}(n) \le \log n$. Let*

$$h_{\mathsf{proof}}(n) := \log T(n) + \Theta(\tau^{-1}\log\log T(n)) - w_{\mathsf{proof}}(n) \qquad \text{and}$$
$$h_{\mathsf{input}}(n) := \lceil \log n \rceil - w_{\mathsf{input}}(n).$$

Moreover, suppose that for some large enough constant $C \ge 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \le 1 - \frac{C\log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$, $H_{\mathsf{proof}}(n) := 2^{h_{\mathsf{proof}}(n)}$, $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$, and $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$. Then $\mathsf{NTIME}_{\mathrm{TM}}[T(n)]$ has a τ-almost rectangular PCPP with an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input oracle and an $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof oracle, whose other parameters are specified in Table 1.

Moreover, V_{row} and V_{col} are projections over seed.row *and* seed.col *respectively, computable in polynomial time given* seed.shared*.*

3 Circuit Avoidance via Hamming Weight Estimation

Recall that for a string x, $\delta(x)$ denotes its relative Hamming weight. Let $\mathscr{C}$ be a multi-output circuit class of stretch $\ell = \ell(n)$, we define the problem $\mathscr{C}$-HammingHit:

Definition 3.1. Let Prep be a preprocessing algorithm, Query be a query algorithm, and $\epsilon > 0$. We say (Prep, Query) is an $\mathsf{FP}^{\mathsf{NP}}$ data structure for $\mathscr{C}$-HammingHit with error $1-\epsilon$, if the following are true:

(Preprocessing) Let $C : \{0,1\}^n \to \{0,1\}^\ell$ be a $\mathscr{C}$ circuit and $\langle C \rangle$ be its description, $\mathsf{Prep}(\langle C \rangle)$ runs in deterministic polynomial time with access to an NP oracle, and produces a string DS of length $\mathrm{poly}(\ell)$.

(Query) For every query $x \in \{0,1\}^n$, $\mathsf{Query}(x)$ runs in deterministic $\ell/\log^{\omega(1)}\ell$ time with random access to DS and outputs a bit b. For every query $x \in \{0,1\}^n$, if $\delta(C(x)) = 1$ then $\mathsf{Query}(x) = 1$; while if $\delta(C(x)) \le \epsilon$ then $\mathsf{Query}(x) = 0$.

Note that the above definition allows polynomial preprocessing time (with an NP oracle) and *non-trivial* query time (i.e., $\ell/\log^{\omega(1)}\ell$).

In this section, we prove our main technical theorem: to solve $\mathscr{C}$-Avoid in $\mathsf{FP}^{\mathsf{NP}}$, it suffices to design an $\mathsf{FP}^{\mathsf{NP}}$ data structure for $\mathscr{C}'$-HammingHit, where $\mathscr{C}' := \mathsf{NC}^0 \circ \mathscr{C}$.

Theorem 3.2. *For every constant $\epsilon > 0$, there is a constant $d > 0$ such that the following holds. Let $\mathscr{C}$ be a circuit class, and $\ell = \ell(n)$ be a good function such that $\ell(n) > n^\gamma$ for some constant $\gamma > 1$. Suppose that for some constant $\eta < \gamma/8$, there is an $\mathsf{FP}^{\mathsf{NP}}$ data structure for $\mathscr{C}'$-$\mathsf{HammingHit}$ with error $1 - \epsilon$, where $\mathscr{C}'$ is the class of $\mathsf{NC}^0_d \circ \mathscr{C}$ circuits with $n \cdot \mathrm{polylog}(\ell)$ inputs and $\ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$ outputs. Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid with stretch $\ell(n)$.*

3.1 Proof of Theorem 3.2

Let $\epsilon > 0$ be some constant, and d be a large constant that depends on ϵ.[18] Assume that there is an $\mathsf{FP}^{\mathsf{NP}}$ data structure $(\mathsf{Prep}, \mathsf{Query})$ for $\mathscr{C}'$-$\mathsf{HammingHit}$ with error ϵ, where $\mathscr{C}'$ is the class of $\mathsf{NC}^0_d \circ \mathscr{C}$ circuits from $n \cdot \mathrm{polylog}(\ell)$ inputs to $\ell^{1-\eta}$ outputs.

Set up. Without loss of generality, we may assume that ℓ is a power of 2. Actually, let ℓ' be the largest power of 2 that is $\le \ell$, we can discard all but the first ℓ' output gates of the input circuit C.

Let $c > 0$ be a constant such that the length of the data structure outputted by $\mathsf{Prep}(\langle C' \rangle)$ is always at most ℓ^c, where C' is any circuit in $\mathscr{C}'$. Let

$$\begin{aligned} \tau &:= 4\eta/(c+2), \\ w_{\mathsf{proof}} &:= \log \ell, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = \ell, \\ h_{\mathsf{proof}} &:= (c+1)\log \ell, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = \ell^{c+1}, \\ N &:= 10 H_{\mathsf{proof}} \cdot n, \\ T &:= H_{\mathsf{proof}} \cdot W_{\mathsf{proof}} / \mathrm{polylog}(\ell). \end{aligned}$$

As long as $\ell \ge n \cdot \mathrm{polylog}(n)$, it follows that $T \ge N \cdot \mathrm{polylog}(N)$. Thus we can use Theorem 2.3 to construct a language

$$L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathrm{TM}}[T] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T/\mathrm{polylog}(T), N/10]_{/(N/10)}.$$

Next we will construct an RTM called M^{PCPP} that attempts to solve L^{hard} in nondeterministic time $T/\mathrm{polylog}(T)$ with $N/10$ nondeterministic guess bits and $N/10$ advice bits. Note that by the time hierarchy theorem, M^{PCPP} has to fail to compute L^{hard} on some input. Our plan is to utilise the failure of M^{PCPP} to find a non-output of C.

Before describing M^{PCPP}, we need the rectangular PCPP (Theorem 2.9) for the following language

$$L^{\mathsf{enc}} := \{\mathsf{Enc}(x) : x \in L^{\mathsf{hard}}\},$$

where we fix an error correcting code $(\mathsf{Enc}, \mathsf{Dec})$ as in Theorem 2.2. Suppose that a string of length N is encoded (via Enc) into a string of length $\tilde{N} = O(N)$. We set the following parameters:

$$\begin{aligned} h_{\mathsf{input}} &:= \left(1 - \frac{\Theta(\log\log T)}{\log T}\right) h_{\mathsf{proof}}, & H_{\mathsf{input}} &:= 2^{h_{\mathsf{input}}} = H_{\mathsf{proof}}/\mathrm{polylog}(\ell), \\ w_{\mathsf{input}} &:= \lceil \log \tilde{N} \rceil - h_{\mathsf{input}}, & W_{\mathsf{input}} &:= 2^{w_{\mathsf{input}}} = n \cdot \mathrm{polylog}(\ell). \end{aligned}$$

By Theorem 2.9, there is a PCPP verifier VPCPP for L^{enc} with oracle access to $\Pi := \mathsf{Enc}(x) \circ \pi$, tosses $r := \log T + O(\tau^{-1} \log\log T)$ random coins, makes $q := O(1)$ queries to Π, and either accepts

[18] Here, d depends on the query complexity of the rectangular PCPP in Theorem 2.9 with soundness ϵ. It follows from the composition theorem (Section 7.2) that d does not depend on τ. Actually, d is the query complexity of the *inner PCPP* ([Mie09]) with soundness ϵ, which is independent from τ.

or rejects. Recall that ϵ is one minus the error tolerance of our HammingHit data structure, and let δ be the relative distance of the code in Theorem 2.2. The PCPP verifier has soundness error ϵ, proximity parameter δ, proof oracle size $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$, input oracle size $H_{\mathsf{input}} \times W_{\mathsf{input}}$, and is τ-almost rectangular. We can verify that the technical conditions in Theorem 2.9 are indeed true. In particular:

- $$\begin{aligned} \tau \log T(n) &\leq \tau(w_{\mathsf{proof}}(n) + h_{\mathsf{proof}}(n)) \\ &\leq \tau(c+2)\log \ell \leq \log \ell = w_{\mathsf{proof}}(n); \end{aligned}$$
- $$\begin{aligned} \frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)} &\leq \frac{\log N - (1 - \Theta(\log\log \ell / \log \ell))(c+1)\log \ell + O(1)}{\log \ell} \\ &\leq \frac{\log n + O(\log\log \ell)}{\log \ell} < 1 - \Omega(1). \end{aligned}$$

3.1.1 The Speed-Up Algorithm M^{PCPP}

Now we define the algorithm M^{PCPP} that attempts to compute L^{hard}. We say an $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix π is *easy* w.r.t. C, if every row of M is in the range of C. The algorithm makes the following "easy-witness" assumption, namely that every $x \in L^{\mathsf{hard}}$ has a PCPP proof which is easy w.r.t. C. We will see that this assumption is the only reason that M^{PCPP} computes L^{hard} incorrectly.

Assumption 3.3. For every $x \in L^{\mathsf{hard}}$, there is a proof matrix $\pi \in \{0,1\}^{H_{\mathsf{proof}} \times W_{\mathsf{proof}}}$ such that:

(Completeness) For every $\mathsf{seed} \in \{0,1\}^r$, $\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})$ accepts.

(Easiness) For every row π_i of π, there exists a string w_i such that $\pi_i = C(w_i)$.

On input $x \in \{0,1\}^N$, we guess H_{proof} strings $w_1, w_2, \ldots, w_{H_{\mathsf{proof}}} \in \{0,1\}^n$. Let π be the $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ proof matrix where for each $i \in [H_{\mathsf{proof}}]$, the i-th row of π is equal to $C(w_i)$. Let $\Pi := \mathsf{Enc}(x) \circ \pi$, and recall that the input oracle (i.e., $\mathsf{Enc}(x)$) is an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ matrix. Now, our goal is to estimate the following quantity in $T/\log^{\omega(1)} T$ time (note that we do not even have time to construct Π explicitly):

$$p_{\mathsf{acc}} := \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[\mathsf{VPCPP}^{\Pi}(\mathsf{seed}) \text{ accepts}].$$

Actually, it suffices to distinguish between the case that $p_{\mathsf{acc}} = 1$ and the case that $p_{\mathsf{acc}} \leq \epsilon$.

We enumerate $\mathsf{seed.shared}$. Let

$$\begin{aligned} (\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) &:= V_{\mathsf{type}}(\mathsf{seed.shared}), \\ (\mathsf{icol}[1], \mathsf{icol}[2], \ldots, \mathsf{icol}[q]) &:= V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared}). \end{aligned}$$

Since $\mathsf{seed.shared}$ is fixed now, $\mathsf{itype}[\cdot]$ is also fixed, and $\mathsf{icol}[\cdot]$ is a function of $\mathsf{seed.col}$.

Consider the following circuit $C' := C'_{\mathsf{seed.shared}}$. It receives as inputs q strings $a_1, a_2, \ldots, a_q$, where for each $j \in [q]$:

- if $\mathsf{itype}[j] = \mathsf{input}$, then a_j is interpreted as a row of the input matrix, and we use $(a_j)_{col}$ to denote the col-th bit of a_j;
- if $\mathsf{itype}[j] = \mathsf{proof}$, then a_j is interpreted as a "seed" such that $C(a_j)$ is a row of the proof matrix, and we use $(a_j)_{col}$ to denote the col-th bit of $\underline{C(a_j)}$. (NOT the col-th bit of a_j!)

In addition, it also receives q bits $pc_1, pc_2, \ldots, pc_q$, representing the contribution of $\mathsf{seed.row}$ in the q parity-check bits.

The circuit C' has $2^{r_{\mathsf{col}}}$ outputs. For each string $\mathsf{seed.col} \in \{0,1\}^{r_{\mathsf{col}}}$, we interpret $\mathsf{seed.col}$ as a number in $[2^{r_{\mathsf{col}}}]$, and the $\mathsf{seed.col}$-th output of C' is

$$\mathsf{VDec}\Big((a_1)_{\mathsf{icol}[1]}, (a_2)_{\mathsf{icol}[2]}, \ldots, (a_q)_{\mathsf{icol}[q]}, pc_1 \oplus pc_1^{\mathsf{col}}, pc_2 \oplus pc_2^{\mathsf{col}}, \ldots, pc_q \oplus pc_q^{\mathsf{col}}\Big).$$

Here, VDec is the decision predicate of VPCPP and pc_i^{col} represents the contribution of $\mathsf{seed.col}$ in the i-th parity-check bit. Note that as $\mathsf{seed.shared}$ is fixed, VDec is also fixed and its only dependencies on $\mathsf{seed.row}$ or $\mathsf{seed.col}$ are via the parity-check bits. It is easy to see that C' only depends on C, the verifier VPCPP, and the shared randomness $\mathsf{seed.shared}$. Let $d := 2q$, then C' is an $\mathsf{NC}^0_d \circ \mathscr{C}$ circuit of description length $O(q|\langle C \rangle|)$ that has

$$O(q \cdot (W_{\mathsf{input}} + n)) \leq n \cdot \mathrm{polylog}(\ell)$$

inputs and

$$2^{|\mathsf{seed.col}|} = \ell / T^{0.25\tau} = \ell^{1-\eta}\mathrm{polylog}(\ell).$$

outputs.

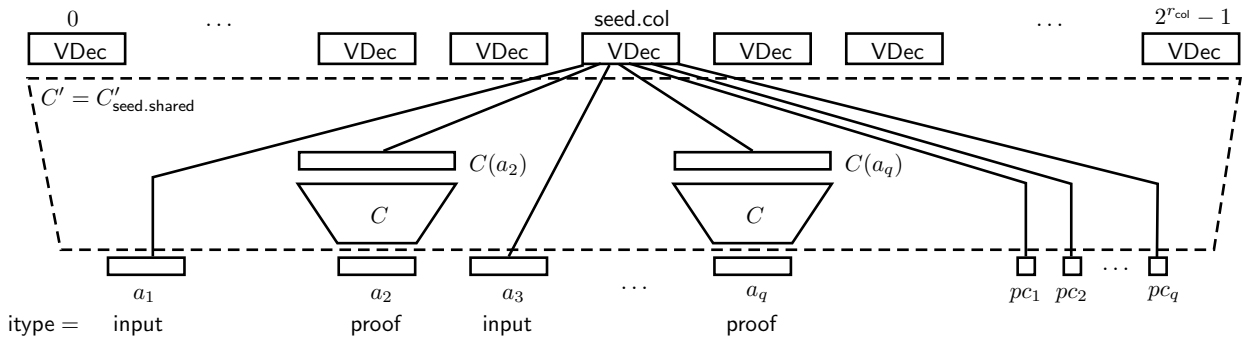

Figure 1: The construction of C'. It is not hard to see that if $C \in \mathscr{C}$, then $C' \in \mathsf{NC}^0_d \circ \mathscr{C}$.

After constructing C', we enumerate $\mathsf{seed.row}$ and compute

$$(\mathsf{irow}[1], \mathsf{irow}[2], \ldots, \mathsf{irow}[q]) := V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared}),$$

as well as the contribution of $\mathsf{seed.row}$ to the parity-check bits $pc_1^{\mathsf{row}}, pc_2^{\mathsf{row}}, \ldots, pc_q^{\mathsf{row}}$.

Let $\tilde{x}_j$ be the j-th row of $\mathsf{Enc}(x)$ (viewed as an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ matrix). For every $j \in [q]$, let

$$a_j := \begin{cases} \tilde{x}_{\mathsf{irow}[j]} & \text{if } \mathsf{itype}[j] = \mathsf{input} \\ w_{\mathsf{irow}[j]} & \text{if } \mathsf{itype}[j] = \mathsf{proof} \end{cases}.$$

We feed the following input to C':

$$\mathsf{input} := (a_1, a_2, \ldots, a_q, pc_1^{\mathsf{row}}, pc_2^{\mathsf{row}}, \ldots, pc_q^{\mathsf{row}}).$$

We can verify that for every $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output bit of $C'(\mathsf{input})$ is exactly equal to $\mathsf{VPCPP}^{\Pi}(\mathsf{seed})$. Thus the relative Hamming weight of $C'(\mathsf{input})$ is exactly the probability that VPCPP accepts, conditioned on $\mathsf{seed.row}$ and $\mathsf{seed.shared}$.

Let $\mathsf{DS} := \mathsf{DS}_{\mathsf{seed.shared}}$ be the data structure returned by $\mathsf{Prep}(C'_{\mathsf{seed.shared}})$. We hardwire DS in our advice.[19] We can use $\mathsf{Query}^{\mathsf{DS}}(\mathsf{input})$ to distinguish between $\delta(C'(\mathsf{input})) = 1$ and $\delta(C'(\mathsf{input})) \leq \epsilon$ in time

$$2^{|\mathsf{seed.col}|}/|\mathsf{seed.col}|^{\omega(1)} = 2^{|\mathsf{seed.col}|}/\ell^{\omega(1)}.$$

If the query algorithm returns 0 for some $\mathsf{seed.shared}$, then we reject x; if it always returns 1, then we accept x.

To summarise, the machine M^{PCPP} runs as follows:

Algorithm 1 A machine M^{PCPP} that attempts to speed up L^{hard}

Input: x; advice $\mathsf{DS}_{\mathsf{seed.shared}}$ for every string $\mathsf{seed.shared} \in \{0,1\}^{|\mathsf{seed.shared}|}$

1: Guess $w_1, w_2, \ldots, w_{H_{\mathsf{proof}}} \in \{0,1\}^n$
2: Compute $(\tilde{x}_1, \ldots, \tilde{x}_{H_{\mathsf{input}}}) \leftarrow \mathsf{Enc}(x)$
3: **for** every string $\mathsf{seed.shared}$ **do**
4: Let $\mathsf{DS} \leftarrow \mathsf{DS}_{\mathsf{seed.shared}}$ be the $\mathsf{HammingHit}$ data structure for the circuit $C'_{\mathsf{seed.shared}}$
5: **for** every string $\mathsf{seed.row}$ **do**
6: $\mathsf{input} \leftarrow \left(a_1, a_2, \ldots, a_q, pc_1^{\mathsf{row}}, pc_2^{\mathsf{row}}, \ldots, pc_q^{\mathsf{row}}\right)$
7: **if** $\mathsf{Query}^{\mathsf{DS}}(\mathsf{input}) = 0$ **then return** reject
8: **return** accept

3.1.2 Analysis of M^{PCPP}

(Time Complexity) Given x, we can compute $\mathsf{Enc}(x)$ in linear time which is not the bottleneck. For each $\mathsf{seed.shared}$ and $\mathsf{seed.row}$ we need to invoke Query once, so the time complexity of M^{PCPP} is

$$\underbrace{2^{|\mathsf{seed.shared}|}}_{\mathsf{seed.shared}} \cdot \underbrace{2^{|\mathsf{seed.row}|}}_{\mathsf{seed.row}} \cdot \underbrace{2^{|\mathsf{seed.col}|}/\log^{\omega(1)}\ell}_{\mathsf{Query}} = H_{\mathsf{proof}} W_{\mathsf{proof}}/\log^{\omega(1)}(W_{\mathsf{proof}}) \leq T/\log^{\omega(1)} T.$$

(Nondeterminism) M^{PCPP} only guesses the strings $w_1, w_2, \ldots, w_{H_{\mathsf{proof}}}$. Therefore, the number of nondeterministic bits it uses is at most $H_{\mathsf{proof}} \cdot n \leq N/10$.

(Advice Complexity) For each $\mathsf{seed.shared}$, we hardcode $\mathsf{Prep}(C'_{\mathsf{seed.shared}})$ which is a length-P string as advice. Therefore, M^{PCPP} requires $(H_{\mathsf{proof}} \cdot W_{\mathsf{proof}})^\tau \cdot \ell^c \leq N/10$ advice bits.

Therefore, M^{PCPP} computes a language in $\mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T/\log^{\omega(1)} T, N/10]_{/(N/10)}$. By Theorem 2.3, there is some input $x_{\mathsf{hard}} \in \{0,1\}^N$ such that $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) \neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$.

Claim 3.4. *For every $x \in \{0,1\}^N$, if $x \notin L^{\mathsf{hard}}$, then M^{PCPP} rejects x.*

Proof. Since $x \notin L^{\mathsf{hard}}$, $\mathsf{Enc}(x)$ is δ-far from $L^{\mathsf{enc}} \cap \{0,1\}^{\tilde{N}}$. For every proof matrix π, the probability over a random $\mathsf{seed} \leftarrow \{0,1\}^r$ that $\mathsf{VPCPP}^{\Pi}(\mathsf{seed})$ accepts is at most ϵ, where $\Pi := \mathsf{Enc}(x) \circ \pi$. This is of course also true if we restrict π to be the matrix whose i-th row is $C(w_i)$. By averaging, there is a setting of $\mathsf{seed.shared}$ and $\mathsf{seed.row}$ such that

$$\Pr_{\mathsf{seed.col}}\left[\mathsf{VPCPP}^{\Pi}(\mathsf{seed}) \text{ accepts}\right] \leq \epsilon.$$

Thus M^{PCPP} will reject in Line 7 of Algorithm 1. □

[19] We do not need to construct either C or any $C'_{\mathsf{seed.shared}}$ inside M^{PCPP} as we can directly use the advice. However, as we will see later, our $\mathsf{FP}^{\mathsf{NP}}$ algorithm for Avoid needs to compute this advice *outside* M^{PCPP}.

It follows from the above claim that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but M^{PCPP} rejects x_{hard}. Now we claim that every PCPP proof for x_{hard} is "hard" against C.

Claim 3.5. *For any proof matrix π such that*

$$\Pr_{\mathsf{seed}}\left[\mathsf{VPCPP}^{\Pi}(\mathsf{seed}) \text{ accepts}\right] = 1,$$

where $\Pi := \mathsf{Enc}(x_{\mathsf{hard}}) \circ \pi$, there is a row of π which is not in the range of C.

Proof. Suppose the claim is false. Then there are strings $w_1, w_2, \ldots, w_{H_{\mathsf{proof}}} \in \{0,1\}^n$ such that the concatenation of $\{C(w_i)\}_i$, denoted as π, is a valid PCPP proof for the assertion

$$\mathsf{Enc}(x) \in L^{\mathsf{enc}}.$$

For every possible seed, $\mathsf{VPCPP}^{\Pi}(\mathsf{seed})$ accepts. It follows that when M^{PCPP} guesses $w_1, w_2, \ldots, w_{H_{\mathsf{proof}}}$, the query algorithm in Line 7 of Algorithm 1 will always return 1. Therefore M^{PCPP} accepts x_{hard}, a contradiction. □

3.1.3 The $\mathsf{FP}^{\mathsf{NP}}$ Range Avoidance Algorithm

Now we present the $\mathsf{FP}^{\mathsf{NP}}$ algorithm for finding a non-output of C.

First, we fix the PCPP verifier VPCPP for L^{enc}. We enumerate every $\mathsf{seed.shared}$ and compute the circuit $C'_{\mathsf{seed.shared}}$. Then we compute $\mathsf{DS}_{\mathsf{seed.shared}} := \mathsf{Prep}(C'_{\mathsf{seed.shared}})$ in $\mathsf{FP}^{\mathsf{NP}}$, concatenate all these data structures $\{\mathsf{DS}_{\mathsf{seed.shared}}\}$ together to form an advice string α. We also compute the code of M^{PCPP} as specified in Algorithm 1.

Since M^{PCPP} is a nondeterministic RTM that runs in time $T/\log^{\omega(1)} T$, guesses at most $N/10$ nondeterministic bits, and takes at most $N/10$ bits of advice, as discussed above, there is an input $x_{\mathsf{hard}} \in \{0,1\}^N$ such that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but M^{PCPP} rejects x_{hard}. Moreover, x_{hard} can be found by the refuter algorithm $\mathcal{R}(1^N, \langle M^{\mathsf{PCPP}}\rangle, \alpha)$ in Theorem 2.3 in $\mathsf{FP}^{\mathsf{NP}}$.

By Claim 3.5, for every $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix π which is a valid PCPP proof for x_{hard}, there is a row of π which is not in the range of C. We can use the NP oracle to find some valid PCPP proof π for x_{hard} (for example, the lexicographically first one). Then we use the NP oracle to check, for each row, whether it is in the range of C. We output the first row that is not in the range of C. This finishes the description of our $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid.

3.2 A Non-trivial Avoidance Algorithm for De Morgan Formulas

As a straightforward application of Theorem 3.2, we present non-trivial $\mathsf{FP}^{\mathsf{NP}}$ avoidance algorithms for De Morgan formulas. In particular, suppose $F : \{0,1\}^n \to \{0,1\}^\ell$ is a function where each output can be computed by a De Morgan formula of size s, and $\ell \geq n^{\omega(\sqrt{s}\log s)}$, then we can find a non-output of F in deterministic $\mathrm{poly}(\ell)$ time with an NP oracle.

We need that any function computed by a size-s formula has approximate degree $\tilde{O}(\sqrt{s})$:

Theorem 3.6 ([Rei11, Tal17]; See also [KKL+21, Lemma 44])**.** *For any integer $s > 0$ and any $0 < \epsilon < 1$, there exists a deterministic algorithm of running time $s^{O(\sqrt{s}\log s \log(1/\epsilon))}$ that given a De Morgan formula F of size s, outputs an ϵ-approximating polynomial of degree $O(\sqrt{s}\log s \log(1/\epsilon))$ for F. That is, the algorithm outputs a multi-linear polynomial p (as sum of monomials) over the reals such that for every $x \in \{0,1\}^n$, we have $|p(x) - F(x)| \leq \epsilon$.*

The above theorem implies the following avoidance algorithm:

Theorem 3.7. *Let $s = s(n) \le \mathrm{poly}(n)$ be a good function. There is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that satisfies the following.*

(Input) *The algorithm receives a multi-output function $F : \{0,1\}^n \to \{0,1\}^\ell$, where each output is computed by a size-s De Morgan formula over the inputs, and $\ell \ge n^{\omega(\sqrt{s}\log s)}$. (The algorithm is given the description of De Morgan formulas for each output bit.)*

(Output) *The algorithm outputs some string $y \in \{0,1\}^\ell \setminus \mathrm{Range}(F)$.*

Proof. Let $\eta > 0$ be a small enough constant. Let $\mathscr{C}'$ be the class of multi-output functions with input length $\ell_{\mathsf{in}} := n \cdot \mathrm{polylog}(\ell)$ and output length $\ell_{\mathsf{out}} := \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$, where each output bit is computed by a size-$O(s)$ De Morgan formula. By Theorem 3.2, it suffices to design an $\mathsf{FP}^{\mathsf{NP}}$ data structure for $\mathscr{C}'$-HammingHit with error $1 - 1/3$.

Let $C : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ be a multi-output function where every output bit is computed by a formula of size $O(s)$. Let C_i be the single-output function denoting the i-th output bit, and p_i be the polynomial that $1/10$-approximates C_i, i.e., for every input x and every i, $|C_i(x) - p_i(x)| \le 1/10$. Let $\alpha_{S,i}$ be the coefficient of the monomial $x_S := \prod_{j \in S} x_j$ in p_i, then

$$p_i(x) = \sum_{|S| \le O(\sqrt{s}\log s)} \alpha_{S,i} x_S.$$

Now, let

$$p(x) := (1/\ell_{\mathsf{out}}) \sum_{i=1}^{\ell_{\mathsf{out}}} p_i(x).$$

It follows that for every x,

$$|\delta(C(x)) - p(x)| \le 1/10.$$

So it suffices to compute $p(x)$ during each query x. In the preprocessing phase, for every monomial x_S, we also compute

$$\alpha_S := (1/\ell_{\mathsf{out}}) \sum_{i=1}^{\ell_{\mathsf{out}}} \alpha_{S,i}.$$

It is easy to see that $p(x) = \sum_S \alpha_S x_S$.

Our $\mathsf{FP}^{\mathsf{NP}}$ data structure for $\mathscr{C}'$-HammingHit works as follows. During the preprocessing phase, we use Theorem 3.6 to compute $p_1, p_2, \ldots, p_{\ell_{\mathsf{out}}}$, and then compute α_S for each monomial x_S. Given a query $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, if $p(x) \le 1/2$ then we output 0; otherwise we output 1.

The preprocessing algorithm runs in time $\ell_{\mathsf{out}} \cdot s^{O(\sqrt{s}\log s)} + (\ell_{\mathsf{in}})^{O(\sqrt{s}\log s)} \le \mathrm{poly}(\ell_{\mathsf{out}})$ (without using the NP oracle). Each query algorithm runs in time

$$(\ell_{\mathsf{in}})^{O(\sqrt{s}\log s)} \le (n \log \ell)^{O(\sqrt{s}\log s)} \le \ell_{\mathsf{out}} / \log^{\omega(1)} \ell_{\mathsf{out}}.$$

Thus we have a non-trivial data structure for $\mathscr{C}'$-HammingHit; we can use it to solve the avoidance problem for De Morgan formulas in $\mathsf{FP}^{\mathsf{NP}}$. □

3.3 Avoidance Algorithms for NC^0 from CSP Sparsification

Actually, one motivation of this work was to unconditionally solve the avoidance problem for NC^0 circuits (with moderate stretch), that is, to prove the following conjecture:

Conjecture 3.8. *For every constant $c \geq 1$ and $\gamma > 0$, let $\mathscr{C}$ be the class of functions with n input bits and $n^{1+\gamma}$ output bits where each output bit depends on at most c input bits. Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-*Avoid.

We did not manage to prove Conjecture 3.8, but we show that if *CSP sparsifiers* [PZ21] are computable in $\mathsf{FP}^{\mathsf{NP}}$, then Conjecture 3.8 is true. Note that the construction in [PZ21] only uses the hypergraph cut sparsifiers in [BST19] as a black box, and thus is computable in randomised polynomial time. Therefore, an $\mathsf{FP}^{\mathsf{NP}}$ derandomisation of the hypergraph cut sparsifiers implies an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for NC^0-Avoid (with stretch $n^{1+\gamma}$ for any constant $\gamma > 0$).

Definition 3.9 (CSP Sparsifiers)**.** A *k-constraint satisfaction problem* (k-CSP) is specified by a k-uniform directed hypergraph $G = (V, E)$ and a predicate $P : \{0,1\}^k \to \{0,1\}$. Here, we say G is a k-uniform directed hypergraph if every edge in G is an ordered length-k tuple $(v_1, v_2, \ldots, v_k)$ where $v_1, v_2, \ldots, v_k \in V$.

Given an assignment $\alpha : V \to \{0,1\}$, the value of the CSP is defined as

$$\mathsf{value}_{G,P}(\alpha) := \frac{1}{|E|} \sum_{(v_1,v_2,\ldots,v_k)\in E} P(v_1, v_2, \ldots, v_k).$$

Let G be a k-uniform directed hypergraph and $P : \{0,1\}^k \to \{0,1\}$ be a predicate. We say that a sub-hypergraph $G' = (V, E' \subseteq E)$ is a *P-sparsifier* of G with error ϵ, if for every assignment $\alpha : V \to \{0,1\}$,

$$\left|\mathsf{value}_{G',P}(\alpha) - \mathsf{value}_{G,P}(\alpha)\right| \leq \epsilon.$$

Remark 3.10. We note that Definition 3.9 is weaker than [PZ21, Definition 9]. In that definition, the error term is ϵ times a quantity which is related to the "volume" of the subset $\alpha^{-1}(0)$ and upper bounded by a constant. In our definition, the error term is simply ϵ. Of course, using a weaker definition of sparsifiers makes it potentially easier to construct them.

The main result in [PZ21] is a randomised algorithm for constructing CSP sparsifiers:

Theorem 3.11 ([PZ21])**.** *For every constants $k \geq 1$ and $\epsilon > 0$, there is a randomised polynomial-time algorithm that given a k-uniform directed hypergraph G and a predicate $P : \{0,1\}^k \to \{0,1\}$, outputs a P-sparsifier of G with error ϵ using $O\big(n\epsilon^{-2}\log(1/\epsilon)\big)$ hyperedges.*

Theorem 3.12. *Suppose that for every constant $k \geq 1$, there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that given any k-uniform directed hypergraph G and any predicate $P : \{0,1\}^k \to \{0,1\}$, outputs a P-sparsifier of G with error $\epsilon = 0.5$ using $\tilde{O}(n)$ hyperedges.*

Then for every constants $\gamma > 0$ and $c \geq 2$, there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid, where $\mathscr{C}$ is the family of NC^0_c circuits with stretch $n^{1+\gamma}$.

Proof. Let $\epsilon := 0.5$ and $\ell(n) := n^{1+\gamma}$. By Theorem 3.2, for some constant $d \geq 1$, it suffices to design an $\mathsf{FP}^{\mathsf{NP}}$ data structure for the $\mathscr{C}'$-HammingHit problem with error $1 - \epsilon = 0.5$, where $\mathscr{C}'$ is the class of $\mathsf{NC}^0_{c\cdot d}$ circuits with $n_{\mathsf{in}} := n \cdot \mathrm{polylog}(n)$ inputs and $n_{\mathsf{out}} := n^{1+8\gamma/9} \cdot \mathrm{polylog}(n)$ output bits.

Let $C : \{0,1\}^{n_{\mathsf{in}}} \to \{0,1\}^{n_{\mathsf{out}}}$ be an input circuit where each output bit only depends on $k := c \cdot d$ input bits. According to the k-ary Boolean function computed by each output bit, we can group these output bits into $2^k = O(1)$ categories, where each category corresponds to a CSP with a fixed predicate $P : \{0,1\}^k \to \{0,1\}$. We compute an ϵ-additive sparsification for these CSPs and take the union of them. (See [PZ21, Remark 11].) We obtain a smaller NC^0 circuit $C' : \{0,1\}^{n_{\mathsf{in}}} \to \{0,1\}^{n_{\mathsf{out}}'}$ such that:

- $n_{\mathsf{out}}' \leq \tilde{O}(n_{\mathsf{in}}) \cdot 2^k \leq \tilde{O}(n_{\mathsf{in}})$;
- for every $x \in \{0,1\}^{n_{\mathsf{in}}}$, $|\delta(C(x)) - \delta(C'(x))| \leq \epsilon$.

Our HammingHit data structure proceeds as follows. In the preprocessing phase, we are given the circuit C, and compute the circuit C' using the NP oracle. Given a query $x \in \{0,1\}^{n_{\mathsf{in}}}$, we compute $\delta(C'(x))$ as an estimation of $\delta(C(x))$ in $O(n_{\mathsf{out}}') < n_{\mathsf{out}}/\log^{\omega(1)} n_{\mathsf{out}}$ time. □

Remark 3.13. The notion of *CSP sparsifiers* is a generalisation of hypergraph cut sparsifiers. Let $\mathsf{Cut} : \{0,1\}^k \to \{0,1\}$ be the predicate which is 0 if all k input bits are equal and is 1 otherwise. Then a cut sparsifier of a hypergraph G is simply a Cut-sparsifier of G.

Near-linear size hypergraph cut sparsifiers are computable in randomised polynomial time [KK15, BST19, SY19, CKN20, KKTY21]. Unfortunately, it is unknown if these constructions can be derandomised (even with an NP oracle).

4 E^{NP} Lower Bounds Are Data Structures

In this section, we show that circuit lower bounds for E^{NP} are equivalent to a certain kind of non-trivial derandomisation *with* E^{NP} *preprocessing*.

4.1 Derandomisation with Preprocessing Implies Circuit Lower Bounds

Let $\mathscr{C}$ be a (single-output) circuit class. We define the circuit-analysis problems for $\mathscr{C}$ *with* E^{NP} *preprocessing*:

Definition 4.1. Let Prep be a preprocessing algorithm, Query be a query algorithm, $\mathscr{C}$ be a circuit class, $s(n)$ be a size parameter, and $\epsilon > 0$. We say (Prep, Query) is a CAPP *data structure for* $\mathscr{C}[s(n)]$ *with* E^{NP} *preprocessing* and error ϵ, if the following are true:

- $\mathsf{Prep}(1^n)$ runs in deterministic $2^{O(n)}$ time with access to an NP oracle, and produces a string DS of length $2^{O(n)}$.
- Let C be a $\mathscr{C}$ circuit with n inputs and size $s(n)$. $\mathsf{Query}(\langle C \rangle)$ runs in deterministic $2^n/n^{\omega(1)}$ time with random access to DS and outputs an estimation of $\Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1]$ within an additive error of ϵ.

If the correctness requirement for $\mathsf{Query}(\cdot)$ is replaced by the following, then we say (Prep, Query) is a GapUNSAT *data structure for* $\mathscr{C}[s(n)]$ *with* E^{NP} *preprocessing* and error $1 - \epsilon$:

- If C is unsatisfiable, then $\mathsf{Query}(\langle C \rangle)$ outputs 0; if $\Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1] \geq 1 - \epsilon$, then $\mathsf{Query}(\langle C \rangle)$ outputs 1.

If the requirement for $\mathsf{Query}(\cdot)$ is replaced by the following, then we say (Prep, Query) is a CAPP data structure for $\mathscr{C}[s(n)]$ with E^{NP} preprocessing *and inverse-circuit-size error*:

- $\mathsf{Query}(\langle C \rangle)$ outputs an estimation of $\Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1]$ within an additive error of $1/s(n)$.

Finally, if the data structure is correct only for infinitely many numbers n, then we say the data structure is an *i.o.* CAPP (or *i.o.* GapUNSAT) data structure.

Theorem 4.2. *Let* $\mathscr{C}$ *be a circuit class and* $\mathrm{poly}(n) \leq s(n) \leq 2^{0.01n}$ *be a good function, such that the following technical conditions hold:*

($\mathscr{C}$ is universal) *For every truth table of length 2^k, there is a $\mathscr{C}$ circuit of size $\mathrm{poly}(2^k)$ that computes this truth table.*

($\mathscr{C}$ computes PARITY) *The* PARITY *function can be computed by a $\mathscr{C}$ circuit of size $\mathrm{poly}(n)$.*

For every constant $\epsilon \in (0,1)$, there is a constant $d \geq 2$ such that the following holds. If there is a GapUNSAT *data structure for $\mathsf{NC}^0_d \circ \mathscr{C}$ circuits of size $\mathrm{poly}(s(n))$ with E^{NP} preprocessing and error $1-\epsilon$, then there is a language in E^{NP} without size-$s(n)$ $\mathscr{C}$ circuits on almost every input length.*

One attempt to prove the above theorem is to directly invoke Theorem 3.2. More precisely, we define a multi-output circuit class that consists of only one circuit $\mathsf{TT}_{\mathscr{C}}(n)$ for each integer n: $\mathsf{TT}_{\mathscr{C}}(n)$ receives as input the description of a $\mathscr{C}$ circuit and outputs its truth table. If we can solve the avoidance problem for $\mathsf{TT}_{\mathscr{C}}(n)$, we could obtain a truth table hard for $\mathscr{C}$ circuits.

However, one issue with this approach is that we need to solve the HammingHit problem of $\mathsf{NC}^0 \circ \mathsf{TT}_{\mathscr{C}}(n)$ in order to solve the avoidance problem for $\mathsf{TT}_{\mathscr{C}}(n)$. It seems unclear how to handle $\mathsf{NC}^0 \circ \mathsf{TT}_{\mathscr{C}}(n)$ circuits using only a CAPP data structure for $\mathscr{C}$. Therefore, we need to follow the proof of Theorem 3.2 (instead of treating it as a black box). The crucial step for getting rid of the top NC^0 circuit is that, in our rectangular PCPP construction, V_{row} and V_{col} have small circuit complexity (i.e., they are projections).

Proof of Theorem 4.2. Let $c \geq 1$ be a constant such that $\mathsf{Prep}(1^n)$ always outputs a data structure of length at most 2^{cn}. Suppose that any $\mathscr{C}$ circuit of size $s(n)$ can be described in bit-length $\ell \leq O(s(n)\log s(n))$. In this proof, we set the following parameters:

$$\begin{aligned} \tau &:= 1/(c+1), \\ w_{\mathsf{proof}} &:= n, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = 2^n, \\ h_{\mathsf{proof}} &:= cn, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = 2^{cn}, \\ N &:= 10 H_{\mathsf{proof}} \cdot \ell = 10 \cdot 2^{cn}\ell, \\ T &:= H_{\mathsf{proof}} \cdot W_{\mathsf{proof}}/\mathrm{poly}(n) = 2^{(c+1)n}/\mathrm{poly}(n). \end{aligned}$$

Our goal is to find, in $\mathsf{DTIME}[2^{O(n)}]^{\mathsf{NP}}$, a truth table of length W_{proof} that does not have size-$s(n)$ $\mathscr{C}$ circuits. Let L^{hard} be the language constructed in Theorem 2.3, i.e.,

$$L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathrm{TM}}[T] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T/\mathrm{polylog}(T), N/10]_{/(N/10)}.$$

Now we construct a nondeterministic RTM M^{PCPP} that attempts to solve L^{hard}. Let $L^{\mathsf{enc}} := \{\mathsf{Enc}(x) : x \in L^{\mathsf{hard}}\}$ where Enc is the error correcting code specified in Theorem 2.2. Suppose that the encoding of length-N strings have length $\tilde{N} = O(N)$. Let

$$\begin{aligned} h_{\mathsf{input}} &:= \left(1 - \frac{\Theta(\log\log T)}{\log T}\right) h_{\mathsf{proof}}, & H_{\mathsf{input}} &:= 2^{h_{\mathsf{input}}} = 2^{cn}/\mathrm{poly}(n), \\ w_{\mathsf{input}} &:= \lceil \log \tilde{N} \rceil - h_{\mathsf{input}}, & W_{\mathsf{input}} &:= 2^{w_{\mathsf{input}}} = \ell \cdot \mathrm{poly}(n). \end{aligned}$$

Let VPCPP be the τ-almost rectangular PCPP verifier for L^{enc} with soundness ϵ specified in Theorem 2.9, where the proof oracle π is an $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix and the input oracle is an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ matrix. We verify the technical conditions of Theorem 2.9 hold:

- $\tau \log T \leq \tau(c+1)n = w_{\mathsf{proof}}$;
- $\dfrac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)} \leq \dfrac{\log N - cn + O(\log n)}{n} \leq \dfrac{\log \ell + O(\log n)}{n} < 1 - \Omega(1)$.

The speed-up machine M^{PCPP} assumes that every row of the proof oracle is the truth table of some circuit in $\mathscr{C}[s(n)]$. It guesses H_{proof} size-$s(n)$ $\mathscr{C}$ circuits $C_1, C_2, \ldots, C_{H_{\mathsf{proof}}}$ such that the i-th row of the proof matrix π is the truth table of C_i. Let $\Pi := \mathsf{Enc}(x) \circ \pi$, now it remains to estimate

$$p_{\mathsf{acc}} := \Pr_{\mathsf{seed}}[\mathsf{VPCPP}^{\Pi}(\mathsf{seed}) \text{ accepts}].$$

We first enumerate $\mathsf{seed.shared}$ and $\mathsf{seed.row}$. Recall that we define

$$\begin{aligned}(\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) &:= V_{\mathsf{type}}(\mathsf{seed.shared}),\\ (\mathsf{irow}[1], \mathsf{irow}[2], \ldots, \mathsf{irow}[q]) &:= V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared}), \text{ and}\\ (\mathsf{icol}[1], \mathsf{icol}[2], \ldots, \mathsf{icol}[q]) &:= V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared}).\end{aligned}$$

(Note that as we only enumerated $\mathsf{seed.shared}$ and $\mathsf{seed.row}$, $\mathsf{itype}[i]$ and $\mathsf{irow}[i]$ are already fixed, but $\mathsf{icol}[i]$ are functions of $\mathsf{seed.col}$.) We need to estimate

$$p_{\mathsf{acc}}(\mathsf{seed.row}, \mathsf{seed.shared}) := \Pr_{\mathsf{seed.col}}[\mathsf{VPCPP}^{\Pi}(\mathsf{seed}) \text{ accepts}]. \tag{2}$$

Now, we create the following circuit $C' := C'_{\mathsf{seed.row},\mathsf{seed.shared}}$ that takes $\mathsf{seed.col}$ as input and accepts if and only if $\mathsf{VPCPP}^{\Pi}(\mathsf{seed})$ accepts. For every $i \in [q]$:

- Suppose $\mathsf{itype}[i] = \mathsf{proof}$. Let D_i be the circuit such that $D_i(\mathsf{seed.col}) = C_{\mathsf{irow}[i]}(\mathsf{icol}[i])$. Since $\mathsf{icol}[i]$ is a projection over $\mathsf{seed.col}$ and $\mathscr{C}$ is closed under projections, D_i can be computed by a $\mathscr{C}$ circuit of size $\mathrm{poly}(\ell)$.
- Suppose $\mathsf{itype}[i] = \mathsf{input}$. Let $D'_{\mathsf{irow}[i]}$ be the $\mathscr{C}$ circuit whose truth table is the $\mathsf{irow}[i]$-th row of the input matrix $\mathsf{Enc}(x)$. Since $\mathscr{C}$ is universal, the size of $D'_{\mathsf{irow}[i]}$ is at most $\mathrm{poly}(W_{\mathsf{input}}) \leq \mathrm{poly}(\ell)$.[20] Let D_i be the circuit such that $D_i(\mathsf{seed.col}) := D'_{\mathsf{irow}[i]}(\mathsf{icol}[i])$, then D_i can also be computed by a $\mathscr{C}$ circuit of size $\mathrm{poly}(\ell)$.

We also construct a circuit $PC_i(\mathsf{seed.col})$ to compute the i-th parity-check bit. In particular, as we have already fixed $\mathsf{seed.row}$ and $\mathsf{seed.shared}$, the i-th parity-check bit is simply the XOR of a subset of indices in $\mathsf{seed.col}$. Since PARITY can be computed by a $\mathscr{C}$ circuit of polynomial size[21], it follows that PC_i is also a $\mathscr{C}$ circuit of size $\mathrm{poly}(|\mathsf{seed.col}|) \leq \mathrm{poly}(n)$.

Let VDec be the decision predicate of VPCPP. The circuit C' is simply

$$C'(\mathsf{seed.col}) := \mathsf{VDec}(D_1(\mathsf{seed.col}), \ldots, D_q(\mathsf{seed.col}), PC_1(\mathsf{seed.col}), \ldots, PC_q(\mathsf{seed.col})).$$

Let $d := 2q$. We can see that C' is an $\mathsf{NC}^0_d \circ \mathscr{C}$ circuit of size $\mathrm{poly}(s(n))$. Therefore, we can use the $\mathsf{GapUNSAT}$ data structure to estimate Eq. (2) in time $2^{r_{\mathsf{col}}}/(r_{\mathsf{col}})^{\omega(1)}$, where $r_{\mathsf{col}} := |\mathsf{seed.col}| = w_{\mathsf{proof}} - (\tau/4)\log T \geq \Omega(n)$. It follows that the above time bound is also $2^{r_{\mathsf{col}}}/n^{\omega(1)}$.

The total running time of M^{PCPP} is thus $2^{|\mathsf{seed}|}/n^{\omega(1)} < T/\log^{\omega(1)} T$. The number of nondeterministic bits that M^{PCPP} guesses is $H_{\mathsf{proof}} \cdot \ell \leq N/10$. The number of advice bits that M^{PCPP} uses is $2^{cn} \leq N/10$. Therefore, M^{PCPP} accepts a language in $\mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T/\log^{\omega(1)} T, N/10]_{/(N/10)}$.

It follows from the definition of L^{hard} that M^{PCPP} fails to compute L^{hard} on some input of length N, for every large enough N. We use the P^{NP} refuter to find an input x_{hard} where $L^{\mathsf{hard}}(x_{\mathsf{hard}}) \neq$

[20] The circuit D'_i may not be uniform (i.e., computable in polynomial time). However, given a circuit $\tilde{D}_i$, it only takes deterministic $\mathrm{poly}(W_{\mathsf{input}}) < 2^{O(n)}$ time to verify whether it computes a given truth table of length W_{input}. Therefore we can use the NP oracle and generate D'_i in $\mathsf{TIME}[\mathrm{poly}(W_{\mathsf{input}})]^{\mathsf{NP}}$.

[21] The circuit for PARITY can be generated in a similar way as Footnote 20. The circuit only takes $|\mathsf{seed.col}| \leq O(n)$ inputs, so it can be verified in deterministic $2^{O(n)}$ time, thus can be generated in $\mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$.

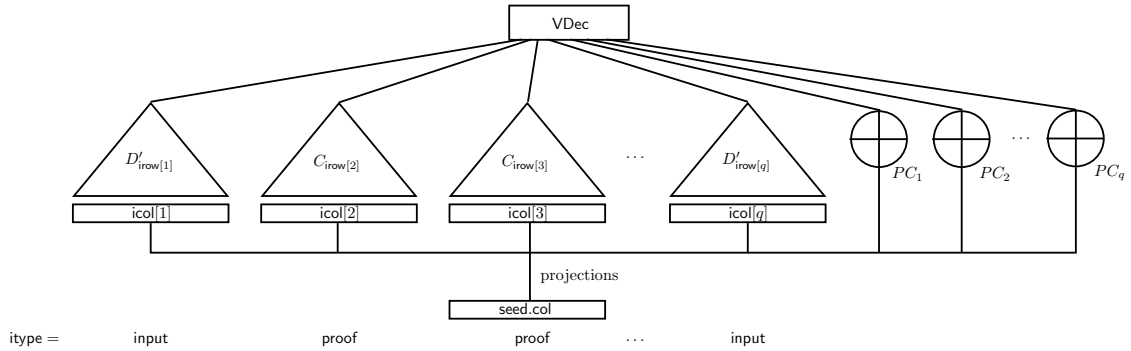

Figure 2: The circuit C'. It is easy to see that $C' \in \mathsf{NC}^0_d \circ \mathscr{C}$.

$M^{\mathsf{PCPP}}(x_{\mathsf{hard}})$. By the construction of M^{PCPP}, it has to be the case that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) = 0$. In this case, for any valid PCPP proof π of "$\mathsf{Enc}(x_{\mathsf{hard}}) \in L^{\mathsf{enc}}$", if we treat π as an $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix, then there has to be some row of π which is not the truth table of any size-$s(n)$ $\mathscr{C}$ circuit. We simply find a valid PCPP proof π and output the row of π that does not have size-$s(n)$ $\mathscr{C}$ circuits. This is easily done in $\mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$. □

To show equivalences in Section 4.3, we also need an infinitely-often version of the above theorem.

Corollary 4.3. *Let $\mathscr{C}$ be a circuit class that is universal and computes* PARITY. *Let* $\mathrm{poly}(n) \leq s(n) \leq 2^{0.01n}$ *be a good function.*

For every constant $\epsilon \in (0,1)$, there is a constant $d \geq 2$ such that the following holds. If there is an i.o. GapUNSAT *data structure for $\mathsf{NC}^0_d \circ \mathscr{C}$ circuits of size $\mathrm{poly}(s(n))$ with E^{NP} preprocessing and error $1-\epsilon$, then there is a language in E^{NP} without size-$s(n)$ $\mathscr{C}$ circuits.*

Proof Sketch. As the proof is almost the same as the proof of Theorem 4.2, we will only sketch the differences here.

We define M^{PCPP} in the same way as Theorem 4.2. Then, M^{PCPP} fails to compute L^{hard} on all large enough input lengths. Note that the correctness of M^{PCPP} on input length n only depends on the correctness of the (i.o.) GapUNSAT data structure on input length

$$r_{\mathsf{col}} = w_{\mathsf{proof}} - (\tau/4)\log T = \frac{3}{4}n + O(\log n).$$

For every integer r_{col} such that the i.o. GapUNSAT data structure is correct, the "easy-witness" assumption fails when $n = \frac{4}{3}r_{\mathsf{col}} - O(\log r_{\mathsf{col}})$. That is, for such integers n, any valid PCPP proof π for $x_{\mathsf{hard}} \in \{0,1\}^n$ contains some row that is not the truth table of any size-$s(n)$ circuit. □

4.1.1 Shaving Logs Implies Lower Bounds, Even with Preprocessing

As a direct corollary of Theorem 4.2, we tighten the connections between circuit lower bounds and non-trivial speed-ups for certain problems in fine-grained complexity. Typically, such a connection states that there is a problem $\mathcal{L}$ for which a $\tilde{O}(n^2)$ algorithm is already known, such that if we could solve $\mathcal{L}$ in $n^2/\log^{\omega(1)} n$ time (i.e., "shaving all logs"), then we could prove a breakthrough circuit lower bound.

Theorem 4.2 implies that we could still obtain the breakthrough lower bound, even if we allow a *preprocessing phase of time n^{100} with an* NP *oracle* before we receive the input of $\mathcal{L}$! For an optimist, this can be seen as an improved approach to prove such lower bounds: Now, we can rely on the

power of P^{NP} preprocessing to solve $\mathcal{L}$, and we (still) only need to obtain a modest improvement over the naïve algorithms.

There are many such connections in the literature, but for illustration, we only consider the following examples:

- LCS (Longest Common Subsequence) over alphabet Σ: Given two sequences $a, b \in \Sigma^N$, find the length of their longest common subsequence.

  Abboud and Bringmann [AB18], improving on [AHWW16], showed that the SAT problem for formulas of size s and n inputs can be reduced to an instance of LCS with two sequences of length $N := 2^{n/2} \cdot s^{1+O(1/\log\log\sigma)}$ over alphabet $[\sigma]$, in $O(N)$ time.

- Closest-LCS-Pair: Given two sets of N length-D strings $\mathcal{A}, \mathcal{B}$, find (or approximate) the maximum length of the longest common subsequence among all pairs $(a, b) \in \mathcal{A} \times \mathcal{B}$.

  Chen, Goldwasser, Lyu, Rothblum, Rubinstein [CGL+19] showed that for every constant $c \geq 1$, the SAT problem for formulas of size s and n inputs can be reduced to an instance of c-approximate Closest-LCS-Pair with $N := 2^{n/2}$ and $D := 2^{\mathrm{polylog}(n)}$.

  > *Proof Sketch in* [CGL+19]. We use Barrington's theorem [Bar89] to transform the formula into a width-5 branching program of size $\mathrm{poly}(s)$. Then we reduce its SAT problem to the following problem (called BP-Satisfying-Pair in [CGL+19]): Given a size-$\mathrm{poly}(s)$ width-5 branching program P on n Boolean inputs, and a set of $N := 2^{n/2}$ strings $\mathcal{A}, \mathcal{B} \subseteq \{0,1\}^n$, determine if there is a pair $(a, b) \in \mathcal{A} \times \mathcal{B}$ such that $P(a, b) = 1$. The next step is to invoke [CGL+19, Theorem 5.6] to reduce BP-Satisfying-Pair to a problem called ϵ-Gap-Max-TropSim, whose input consists of two sets of N tensors of size $D := 2^{O(\log^2 n \log\log n)}$. Finally, this problem reduces to $O(1)$-approximate Closest-LCS-Pair over N length-D strings. □

- $\mathbb{Z}$-OV (Hopcroft's Problem): Given N points $\vec{v}^1, \vec{v}^2, \ldots, \vec{v}^N \in \mathbb{Z}^D$, find an orthogonal pair, i.e., two vectors $\vec{v}^a$ and $\vec{v}^b$ such that

$$\sum_{i=1}^{D} v_i^a \cdot v_i^b = 0.$$

  Chen [Che18] showed that the SAT problem on $\mathsf{THR} \circ \mathsf{THR}$ circuits of size s and n inputs can be reduced to $\mathrm{poly}(s)$ $\mathbb{Z}$-OV instances on $N := 2^{n/2}$ vectors of dimension $D := \mathrm{poly}(s)$.

  > *Proof Sketch in* [Che18]. Given a $\mathsf{THR} \circ \mathsf{THR}$ circuit of size s, we use [Che18, Theorem 1.6] to transform it into an $\mathsf{OR} \circ \mathsf{THR} \circ \mathsf{MAJ}$ circuit of size $\mathrm{poly}(s)$, then use [HP10] to transform it into an $\mathsf{OR} \circ \mathsf{ETHR} \circ \mathsf{MAJ}$ circuit of size $\mathrm{poly}(s)$. Here, ETHR denotes "exact threshold gates", which outputs 1 if a certain linear combination of its input is exactly equal to its threshold parameter, and outputs 0 otherwise; the transformation can be performed in deterministic $\mathrm{poly}(s)$ time. Finally, we reduce the satisfiability of each bottom $\mathsf{ETHR} \circ \mathsf{MAJ}$ circuit to $\mathbb{Z}$-OV. □

The naïve algorithms for LCS, Closest-LCS-Pair, and $\mathbb{Z}$-OV run in $O(N^2)$, $O(N^2 D^2)$, and $O(N^2 D)$ time respectively. The above reductions show that a modest improvement of these quadratic-time algorithms (i.e., "shaving all logs") would imply new SAT algorithms for frontier circuit classes, e.g., NC^1 or $\mathsf{THR} \circ \mathsf{THR}$. By the Algorithmic Method [Wil13, CW19], these SAT algorithms imply long-standing circuit lower bounds for these classes.

To state our corollary, we need the following definition of solving a problem with P^{NP} preprocessing.

Definition 4.4. We say that a problem $\mathcal{L}$ can be solved in $T(n)$ time *with* P^{NP} *preprocessing* if there are two algorithms $(\mathsf{Prep}, \mathsf{Query})$ such that the following holds:

- Prep receives an input 1^n, runs in time $\mathrm{poly}(n)$ with access to an NP oracle, and outputs a string DS of length $\mathrm{poly}(n)$.
- Query receives an input x of $\mathcal{L}$, has random access to DS, runs in time $T(n)$, and correctly decides whether $x \in \mathcal{L}$.

Now we present the following corollary of Theorem 4.2, which states that even if we allow a P^{NP} preprocessing phase (which runs in arbitrary polynomial time in N but does not see the input), such a modest improvement would still imply breakthrough circuit lower bounds:

Corollary 4.5. *The following are true:*

- *Suppose that* LCS *of length-N strings over any constant-size alphabet can be solved in $N^2/\log^{\omega(1)} N$ time, even with* P^{NP} *preprocessing, then* $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{NC}^1$.
- *Suppose there is a constant $c \geq 1$ such that for any $D = 2^{\mathrm{poly}(\log\log N)}$,* $\mathsf{Closest\text{-}LCS\text{-}Pair}$ *of N length-D strings can be c-approximated in $N^2/\log^{\omega(1)} N$ time, even with* P^{NP} *preprocessing, then* $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{NC}^1$.
- *Suppose that for any $D = \mathrm{polylog}(N)$, $\mathbb{Z}$-OV for N points in $\mathbb{Z}^D$ can be solved in $N^2/\log^{\omega(1)} N$ time, even with* P^{NP} *preprocessing, then* $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{THR} \circ \mathsf{THR}$.

Remark 4.6. Since NC^1 satisfies all technical conditions in Theorem 4.2 (NC^1 is universal, computes PARITY efficiently, and is closed under adding NC^0 circuits at the top), the first two items of Corollary 4.5 are straightforward. For $\mathsf{THR} \circ \mathsf{THR}$:

- Since $\mathsf{THR} \circ \mathsf{THR}$ contains CNF (i.e., $\mathsf{AND} \circ \mathsf{OR}$), it is clearly universal.
- There is a polynomial-size $\mathsf{THR} \circ \mathsf{THR}$ circuit that computes PARITY [Mur71, PS94].
- Finally, for $d = O(1)$, the SAT problem (thus, the $\mathsf{GapUNSAT}$ problem) for $\mathsf{NC}^0_d \circ \mathsf{THR} \circ \mathsf{THR}$ reduces to $\mathrm{poly}(n^d)$ instances of the SAT problem for $\mathsf{THR} \circ \mathsf{THR}$.

  More precisely, since $\mathsf{THR} \circ \mathsf{THR}$ is closed under negation, the SAT problem for $\mathsf{NC}^0_d \circ \mathsf{THR} \circ \mathsf{THR}$ reduces to the SAT problem for $2^{O(d)}$ instances of $\mathsf{AND}_d \circ \mathsf{THR} \circ \mathsf{THR}$. Using the fact that $\mathsf{THR} \subseteq \mathsf{OR} \circ \mathsf{ETHR}$ [HP10], we reduce the problem to the SAT problem for $\mathsf{AND}_d \circ \mathsf{OR} \circ \mathsf{ETHR} \circ \mathsf{THR}$. Enumerating the set of d bottom $\mathsf{ETHR} \circ \mathsf{THR}$ circuits that are satisfied, we reduce the problem to $\mathrm{poly}(n)^d \leq \mathrm{poly}(n)$ instances of the SAT problem for $\mathsf{AND}_d \circ \mathsf{ETHR} \circ \mathsf{THR}$ circuits. Since $\mathsf{AND} \circ \mathsf{ETHR} \subseteq \mathsf{ETHR}$ and $\mathsf{ETHR} \circ \mathsf{THR} = \mathsf{ETHR} \circ \mathsf{ETHR} \subseteq \mathsf{THR} \circ \mathsf{THR}$ [HP10], every $\mathsf{AND}_d \circ \mathsf{ETHR} \circ \mathsf{THR}$ circuit is equivalent to a $\mathsf{THR} \circ \mathsf{THR}$ circuit. Finally, notice that every transformation mentioned above can be implemented in deterministic polynomial time.

4.2 Technical Preliminaries

To prove the full equivalence results in Section 4.3, we need the following preliminaries.

4.2.1 Pseudorandom Generators

Let $\mathscr{C}$ be a circuit class, $\epsilon > 0$, and $r(n) < n$ be a good function. A *pseudorandom generator* (PRG) with seed length $r(n)$ that ϵ-fools $\mathscr{C}$ is a function $G : \{0,1\}^r \to \{0,1\}^n$ such that for every circuit $C \in \mathscr{C}$,

$$\left| \Pr_{x \leftarrow \{0,1\}^n}[C(x) = 1] - \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[C(G(\mathsf{seed})) = 1] \right| \leq \epsilon.$$

We also say G is an *i.o. PRG* if the above condition holds for infinitely many lengths n.

In this paper, we will mostly consider E^{NP}-computable PRGs, where G is computable in $2^{O(r)}$ time with access to an NP oracle. (It is without loss of generality to assume that $r \geq \Omega(\log n)$.)

A PRG $G : \{0,1\}^r \to \{0,1\}^n$ is *seed-extending* if for every $\mathsf{seed} \in \{0,1\}^r$, the first r bits of $G(\mathsf{seed})$ is always equal to seed.[22]

We need the classical construction of PRGs from average-case lower bounds [NW94]. Let Junta_k be the class of k-juntas, i.e., functions that only depend on k input bits. We have:

Theorem 4.7 ([NW94], see also [CR20, Theorem 6.4])**.** *Let m, ℓ, a be integers such that $a \leq \ell$, and let $t := O(\ell^2 \cdot m^{1/a}/a)$. Let $\mathscr{C}$ be a circuit class closed under negation. There is a function $G : \{0,1\}^{2^\ell} \times \{0,1\}^t \to \{0,1\}^m$ computable in deterministic $\mathrm{poly}(m, 2^t)$ time such that the following holds.*

For any function $Y : \{0,1\}^\ell \to \{0,1\}$ represented as a length-2^ℓ truth table, if Y cannot be $(1/2+\epsilon/m)$-approximated by $\mathscr{C}[S] \circ \mathsf{Junta}_a$ circuits (i.e., the top $\mathscr{C}$ circuit has size S), then $G(Y, -)$ is a PRG that ϵ-fools every $\mathscr{C}[S]$ circuit. That is, for any circuit $C \in \mathscr{C}[S]$,

$$\left| \Pr_{s \leftarrow \{0,1\}^t}[C(G(Y,s)) = 1] - \Pr_{x \leftarrow \{0,1\}^m}[C(x) = 1] \right| \leq \epsilon.$$

Moreover, for every Y, the function $G(Y, -)$ is seed-extending.

4.2.2 Elementary Properties of Norm and Inner Product

We discuss some properties of norms and the inner product of functions on Boolean cubes, which will be useful for us. For a function $f : \{0,1\}^n \to \mathbb{R}$, we define its ℓ_p-norm as

$$\|f\|_p := \left(\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n} [|f(x)|^p] \right)^{1/p}$$

In particular, the ℓ_∞-norm is defined as the maximum absolute value of f.

$$\|f\|_\infty := \max_{x \in \{0,1\}^n} |f(x)|.$$

For $m \geq 2$ and functions $f_1, f_2, \ldots, f_d : \{0,1\}^n \to \mathbb{R}$, we use the following definition for convenience:

$$\langle f_1, f_2, \ldots, f_d \rangle := \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n} \left[\prod_{i=1}^d f_i(x) \right].$$

We need the following generalization of the Hölder's inequality:

Fact 4.8. *Let d be an integer. For functions $f_1, f_2, \ldots, f_d : \{0,1\}^n \to \mathbb{R}$ we have that*

$$\left\| \prod_{i=1}^d f_i \right\|_1 \leq \prod_{i=1}^d \|f_i\|_d$$

We need the following simple lemma for this paper.

[22] Note that a cryptographic PRG [HILL99], where the PRG itself (G) is more efficient than the adversaries ($\mathscr{C}$), cannot be seed-extending. In contrast, a complexity-theoretic PRG [NW94], where the PRG could take more time to compute than the adversaries have, could be seed-extending. Indeed, the classical Nisan-Wigderson PRG can be made seed-extending.

Lemma 4.9 (a generalisation of [CW19, Lemma 28]). *For any integer $d \geq 2$ and functions $f_1, f_2, \ldots, f_d$ and $g_1, g_2, \ldots, g_d$ from $\{0,1\}^n \to \mathbb{R}$ and $\varepsilon, \alpha > 0$, suppose for all $i \in [d]$ we have:*

- $\|f_i\|_d \leq \alpha$ *and* $\|g_i\|_d \leq \alpha$,
- $\|f_i - g_i\|_d \leq \varepsilon$.

Then $|\langle f_1, f_2, \ldots, f_d\rangle - \langle g_1, g_2, \ldots, g_d\rangle| \leq d \cdot \alpha^{d-1} \cdot \varepsilon$.

Proof. We have

$$\begin{aligned}
|\langle f_1, \ldots, f_d\rangle - \langle g_1, \ldots, g_d\rangle| &\leq \sum_{i=1}^{d} |\langle f_1, \ldots, f_i, g_{i+1}, \ldots, g_d\rangle - \langle f_1, \ldots, f_{i-1}, g_i, \ldots, g_d\rangle| \\
&\leq \sum_{i=1}^{d} |\langle f_1, \ldots, f_i - g_i, g_{i+1}, \ldots, g_d\rangle| \\
&\leq d \cdot \alpha^{d-1} \cdot \varepsilon \text{ (by Fact 4.8).}
\end{aligned}$$

□

4.2.3 Linear Sum of Circuits

$\mathsf{Sum} \circ \mathscr{C}$ circuits. Let $\mathscr{C}$ be a circuit class. A $\mathsf{Sum} \circ \mathscr{C}$ circuit $C : \{0,1\}^n \to \mathbb{R}$ is a circuit of the following form:

$$C(x) = \sum_{i=1}^{\ell} \alpha_i C_i(x),$$

where each $\alpha_i \in \mathbb{R}$ and each C_i is a $\mathscr{C}$ circuit. The *complexity* of C is defined as

$$\mathsf{complexity}(C) := \max\left\{\sum_{i=1}^{\ell} |\alpha_i|, \sum_{i=1}^{\ell} |C_i|\right\}.$$

If for every input $x \in \{0,1\}^n$, we have $C(x) \in [0,1]$, then we say C is an $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit. Recall that the ℓ_1-distance between a $\mathsf{Sum} \circ \mathscr{C}$ circuit C and a function f is

$$\|C - f\|_1 = \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n} |C(x) - f(x)|.$$

We define bin_C as the Boolean function that is closest to C. That is, for every $x \in \{0,1\}^n$, if $C(x) \leq 0.5$ then $\mathsf{bin}_C(x) = 0$, otherwise $\mathsf{bin}_C(x) = 1$.

$\widetilde{\mathsf{Sum}} \circ \mathscr{C}$ circuits. Let $\delta \in [0, 0.5)$. A $\mathsf{Sum} \circ \mathscr{C}$ circuit C is said to be a $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuit, if for every input $x \in \{0,1\}^n$, either $|L(x) - 1| \leq \delta$ or $|L(x)| < \delta$. We say $C(x) = 1$ if $|L(x) - 1| \leq \delta$, and $C(x) = 0$ otherwise.

4.2.4 Algorithms for Linear Sum of Circuits

In the context of the Algorithmic Method, one advantage of $\mathsf{Sum} \circ \mathscr{C}$ circuits is that it preserves *algorithms*: circuit analysis algorithms for $\mathscr{C}$ often imply circuit analysis algorithms for $\mathsf{Sum} \circ \mathscr{C}$. Below are a few examples that will be useful for us.

PRGs fooling $\mathscr{C}$ also fools $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$. Suppose that G is a PRG that fools $\mathscr{C}$ circuits, we show that it also fools $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$ circuits. For completeness, we include a proof in Appendix A.2.

Lemma 4.10. *Let $\epsilon, \delta > 0$, $\mathscr{C}$ be a circuit class, $s(n)$ be a good function, and $G : \{0,1\}^r \to \{0,1\}^n$ be a PRG that ϵ-fools $\mathscr{C}$ circuits of size $s(n)$. For $\epsilon' := 2\delta + \epsilon \cdot s(n)$, G also ϵ'-fools $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuits of complexity $s(n)$.*

CAPP algorithms for $\mathsf{Sum} \circ \mathscr{C}$. Let $d \geq 2$ be a constant. Given a CAPP algorithm for $\mathsf{AND}_d \circ \mathscr{C}$ with inverse-circuit-size error, we can also construct a CAPP algorithm for $\mathsf{AND}_d \circ \mathscr{C}'$ with constant error, where $\mathscr{C}'$ is the class of $\mathsf{Sum} \circ \mathscr{C}$ circuits C' that has ℓ_d-distance at most δ to $\mathsf{bin}_{C'}$.[23] (Recall that $\mathsf{bin}_{C'}$ is the Boolean function closest to C'.) For completeness, we include a proof in Appendix A.3.

Lemma 4.11. *Let $d \geq 2$ be a constant, $s = s(n), T = T(n)$ be good functions, and $\mathscr{C}$ be a circuit class. Suppose that there is a deterministic $T(n)$-time algorithm for the CAPP problem for $\mathsf{AND}_d \circ \mathscr{C}$ circuits with inverse-circuit-size error, where the bottom $\mathscr{C}$ circuits have size $\Theta(s(n)^d)$.*

Then, there is a constant $\delta > 0$ and an algorithm that achieves the following.

(Input) *The algorithm is given d $\mathsf{Sum} \circ \mathscr{C}$ circuits $C_1, C_2, \ldots, C_d$ as inputs, where for each $i \in [d]$, C_i has complexity at most $s(n)$, and it is promised that $\|C_i - \mathsf{bin}_{C_i}\|_d \leq \delta$.*

(Output) *The algorithm estimates the following quantity within an additive error of $(1/6) \cdot 2^{-d}$.*

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n} [\mathsf{bin}_{C_1}(x) \wedge \mathsf{bin}_{C_2}(x) \wedge \cdots \wedge \mathsf{bin}_{C_d}(x)].$$

(Complexity) *The algorithm runs in deterministic $O(s(n)^d \cdot T(n))$ time.*

Verification of $\mathsf{Sum} \circ \mathscr{C}$ circuits. We need the following lemma for testing whether a $\mathsf{Sum} \circ \mathscr{C}$ circuit has low ℓ_d-distance to its closest Boolean function. The lemma has a similar proof to [CR20, Lemma 5.3], for completeness, we include a proof in Appendix A.4.

Lemma 4.12. *Let $S \in \mathbb{N}$ and $d \geq 2$ where d is an even number. Suppose we are given S reals $(\alpha_i)_{i \in [S]}$, S $\mathscr{C}$ circuits $(C_i)_{i \in [S]}$, and a parameter $\delta < 0.01/d$. Let $\alpha_{\mathsf{tot}} := \sum_{i \in [S]} |\alpha_i|$ and $\epsilon := \frac{\delta^d}{2 \cdot 3^d \cdot (\alpha_{\mathsf{tot}}+1)^{2d}}$. Suppose the CAPP problem for $\mathsf{AND}_{2d} \circ \mathscr{C}$ with error ϵ can be solved in $T(n)$ time. Let $f = \sum_{i=1}^{S} \alpha_i \cdot C_i$. There is an algorithm A running in $O(T(n) \cdot (S+1)^{2d})$ time such that:*

- *If one of the following conditions hold, then A always accepts;*
  - $\|f - \mathsf{bin}_f\|_\infty \leq \delta/3$
  - $\|f - \mathsf{bin}_f\|_1 \leq (\delta/3)^d$ *and* $f(x) \in [0,1]$ *for any* $x \in \{0,1\}^n$
- *if $\|f - \mathsf{bin}_f\|_d \geq \delta$, then A always rejects;*
- *otherwise, A can output anything.*

[23] Here, the distance is measured in ℓ_d for the following reason. If the verification algorithm (Lemma 4.12) accepts an input $\mathsf{Sum} \circ \mathscr{C}$ circuit, we only know that this circuit is close to some Boolean function in ℓ_d-distance. Our CAPP algorithm needs to deal with *every* $\mathsf{Sum} \circ \mathscr{C}$ circuit that passes the verification algorithm.

4.2.5 Worst-Case Hardness from PRGs

The following simple fact states that PRGs imply worst-case hardness.

Fact 4.13 ([CLLO21, Proposition 9]). *Let $\mathscr{C}$ be a circuit class and $r = r(n)$ be a good function. Suppose there is an E^{NP}-computable PRG (i.o. PRG respectively) $G : \{0,1\}^r \to \{0,1\}^n$ that ϵ-fools $\mathscr{C}$, where $\epsilon < 1 - 2^{r-n}$. Then there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ that cannot be computed by $\mathscr{C}$ circuits on almost every input length (infinitely many input lengths respectively).*

We also need the following lemma stating that seed-extending PRGs fooling $\mathscr{C}$ imply ℓ_1-distance lower bounds against $[0,1]\text{-}\mathsf{Sum} \circ \mathscr{C}$. For completeness, we include a proof in Appendix A.5.

Lemma 4.14. *Let $s = s(n)$ be a good function, $G : \{0,1\}^{n-1} \to \{0,1\}^n$ be a seed-extending PRG that ϵ-fools $\mathscr{C}$ circuits of size $s(n)$. Then the following problem L has ℓ_1-distance at least $(1/2 - \epsilon \cdot s(n))$ from $[0,1]\text{-}\mathsf{Sum} \circ \mathscr{C}$ circuits of complexity $s(n)$.*

$$L = \big\{y \in \{0,1\}^n : \exists x \in \{0,1\}^{n-1}, s.t.\ G(x) = y\big\}.$$

4.2.6 Hardness Amplification

Hardness amplification with a TC^0 decoder. We need the following result.

Theorem 4.15 ([GR08]). *Let $\epsilon > 2^{-c\sqrt{n}}$ for some absolute constant c. There are two algorithms Amp and Dec such that:*

- *For some constant $d > 1$, Amp takes as input the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ and outputs the truth table of a function $\mathsf{Amp}(f) : \{0,1\}^{dn} \to \{0,1\}$.*
- *$\mathsf{Dec}^{(-)}$ receives an oracle h, an input $x \in \{0,1\}^n$, an advice string $\alpha \in \{0,1\}^{O(\log \epsilon^{-1})}$, as well as two random strings r_1, r_2, and outputs a bit b.*
- *For every function $h : \{0,1\}^{dn} \to \{0,1\}$ that $(1/2+\epsilon)$-approximates $\mathsf{Amp}(f)$,*

$$\Pr_{r_1}\Big[\exists \alpha \in \{0,1\}^{O(\log \epsilon^{-1})} \text{ s.t. } \forall x \in \{0,1\}^n, \Pr_{r_2}[\mathsf{Dec}^h(\alpha, x, r_1, r_2) = f(x)] > 9/10\Big] > 99/100.$$

- *Amp runs in deterministic $2^{O(n)}$ time and Dec is a TC^0 oracle circuit of size $\mathrm{poly}(n, \epsilon^{-1})$.*

A non-standard XOR lemma. For a function $f : \{0,1\}^n \to \{0,1\}$ and an integer k, we define the function $f^{\oplus k}$ to take k inputs $x_1, x_2, \ldots, x_k \in \{0,1\}^n$ and compute

$$f^{\oplus k}(x_1, x_2, \ldots, x_k) = f(x_1) \oplus f(x_2) \oplus \cdots \oplus f(x_k).$$

We need the following XOR lemma with a linear sum corrector.

Theorem 4.16 ([Lev87], [CLW20, Lemma 3.8]). *Let $\mathscr{C}$ be a circuit class that is closed under negation and projection. Let $\delta < 1/2$, $k \in \mathbb{N}$ be a parameter, and*

$$\epsilon_k := (1-\delta)^{k-1}(1/2 - \delta).$$

For any function $f : \{0,1\}^n \to \{0,1\}$, if f cannot be $(1-\delta)$-approximated in ℓ_1-distance by $[0,1]\text{-}\mathsf{Sum} \circ \mathscr{C}$ circuits of complexity $O\Big(\frac{ns}{(\delta \cdot \epsilon_k)^2}\Big)$, then $f^{\oplus k}$ cannot be $(1/2+\epsilon_k)$-approximated by $\mathscr{C}$ circuits of size s.

4.3 Equivalences between Circuit Lower Bounds and Derandomisation with Preprocessing

We show that circuit lower bounds for E^{NP} are actually *equivalent* to derandomisation with E^{NP} preprocessing.

Our first theorem shows that if $\mathscr{C}$ is a "strong enough" circuit class, then the worst-case lower bound $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathscr{C}$ is equivalent to non-trivial derandomisation of $\mathscr{C}$ circuits with E^{NP} preprocessing. For simplicity, we only consider a few typical circuit classes (TC^0, NC^1, and $\mathsf{P}_{/\mathrm{poly}}$), but it is clear from the argument that we only rely on a few closure properties of $\mathscr{C}$.

Theorem 4.17. *Let $\mathscr{C} \in \{\mathsf{TC}^0, \mathsf{NC}^1, \mathsf{P}_{/\mathrm{poly}}\}$. The following are equivalent:*

1. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ that cannot be $(1/2+1/n^k)$-approximated by* i.o.-$\mathscr{C}[n^k]$.
2. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin$* i.o.-$\mathscr{C}[n^k]$.
3. *For every constant $k \geq 1$, there is a $\mathsf{GapUNSAT}$ algorithm for $\mathscr{C}[n^k]$ with E^{NP} preprocessing, error $1 - 0.01$, and $2^n/n^{\omega(1)}$ query time.*
4. *For every constant $\delta > 0$ and every good function $s = s(n)$, there is a CAPP algorithm for $\mathscr{C}[s]$ circuits with $\mathsf{TIME}[\exp(s^\delta)]^{\mathsf{NP}}$ preprocessing, $\exp(s^\delta)$ query time, and inverse-circuit-size error.*
5. *For every constant $k \geq 1$, there is an E^{NP}-computable PRG with seed length $n-1$ that $(1/3)$-fools $\mathscr{C}[n^k]$ circuits.*
6. *For every constant $k \geq 1$, there is an E^{NP}-computable PRG with seed length $n^{1/k}$ that $(1/n^k)$-fools $\mathscr{C}[n^k]$ circuits.*

Proof. (4) $\implies$ (3) and (6) $\implies$ (5) are trivial.

(3) $\implies$ (2) follows from Theorem 4.2 and the fact that $\mathscr{C}$ is "typical". In particular, $\mathscr{C}$ is universal (any Boolean function can be computed by $\mathscr{C}$ circuits of large enough size), PARITY has polynomial-size $\mathscr{C}$ circuits, and $\mathscr{C}$ is closed under composition of an NC^0 circuit at the top.

(2) $\implies$ (1) follows from the locally list-decodable code of [GR08] with TC^0 decoders. In particular, let $\epsilon := 1/n^{k+1}$, and $(\mathsf{Amp}, \mathsf{Dec})$ be the locally list-decodable code specified in Theorem 4.15. Let $f : \{0,1\}^n \to \{0,1\}$ be a hard function in $\mathsf{E}^{\mathsf{NP}} \setminus$ i.o.-$\mathscr{C}[n^{O(k)}]$, and $tt_f \in \{0,1\}^{2^n}$ be its truth table. Let $tt_{f'} := \mathsf{Amp}(tt_f)$, then for some constant $d > 1$, $tt_{f'}$ is the truth table of a function $f' : \{0,1\}^{dn} \to \{0,1\}$ which is computable in E^{NP}.

We claim that f' cannot be $(1/2+1/n^{k+1})$-approximated by $\mathscr{C}$ circuits of size n^{k+1} (thus cannot be $(1/2 + 1/(dn)^k)$-approximated by $\mathscr{C}$ circuits of $(dn)^k$). Suppose for contradiction that there is a circuit $C \in \mathscr{C}[n^{k+1}]$ that $(1/2 + 1/n^{k+1})$-approximates f'. Fix a good r_1, then there is a string $\alpha \in \{0,1\}^{O(\log \epsilon^{-1})}$ such that for every $x \in \{0,1\}^n$,

$$\Pr_{r_2}[\mathsf{Dec}^h(\alpha, x, r_1, r_2) = f(x)] > 9/10.$$

By Adleman's argument [Adl78] over r_2, and recall that $\mathsf{Dec}^{(-)}$ is a TC^0 oracle circuit, it follows that f can be computed by a $\mathscr{C}$ circuit of size $n^{O(k)}$. This contradicts the worst-case hardness of f.

(1) $\implies$ (6) follows from the Nisan-Wigderson generator [NW94]. In particular, we set the following parameters:

$$\ell := n^{1/3k}, m := n, a := \log n, t := O(\ell^2 \cdot m^{1/a}/a) := O(\ell^2) \le n^{1/k}.$$

Let $f : \{0,1\}^\ell \to \{0,1\}$ be a function in E^{NP} that cannot be $(1/2 + 1/n^{k+1})$-approximated by $\mathscr{C}$ circuits of size $n^k \cdot \mathrm{poly}(2^a) \le \mathrm{poly}(n)$. Then it cannot be $(1/2 + 1/n^{k+1})$-approximated by $\mathscr{C} \circ \mathsf{Junta}_a$ circuits where the top $\mathscr{C}$ circuit has size n^k. By Theorem 4.7, there is a function $G : \{0,1\}^{2^\ell} \times \{0,1\}^t \to \{0,1\}^m$ computable in $\mathrm{poly}(m, 2^t) \le 2^{O(t)}$ time, such that $G(tt(f), \mathcal{U}_t)$ $(1/n^k)$-fools every $\mathscr{C}[n^k]$ circuit. Since $f \in \mathsf{E}^{\mathsf{NP}}$, the generator $G(tt(f), \cdot)$ is computable in E^{NP}.

(6) $\implies$ (4): Let $G : \{0,1\}^{s^{\delta/2}} \to \{0,1\}^s$ be the E^{NP}-computable PRG that $(1/s)$-fools $\mathscr{C}[2s]$ circuits with s inputs. (Given a $\mathscr{C}[s]$ circuit C with n inputs, we can pad some dummy inputs to C so that C becomes a $\mathscr{C}$ circuit of size $2s$ with s inputs; C only depends on the first n inputs.)

In the E^{NP} preprocessing phase, we compute the whole PRG (i.e., $G(x)$ for every $x \in \{0,1\}^{s^{\delta/2}}$). Given a circuit $C \in \mathscr{C}[s]$ as a CAPP query, we simply compute

$$\Pr_{x \leftarrow \{0,1\}^{s^{\delta/2}}}[C(G(x)) = 1],$$

which estimates the accept probability of C within additive error $1/s$. The query algorithm runs in $\mathrm{poly}(s) \cdot \exp(s^{\delta/2}) < \exp(s^\delta)$ time.

(5) $\implies$ (2) follows from Fact 4.13. In particular, suppose there is an E^{NP}-computable PRG $G : \{0,1\}^{n-1} \to \{0,1\}^n$ that $(1/3)$-fools $\mathscr{C}[n^k]$ circuits. Since $1/3 < 2^{(n-1)-n} = 1/2$, it follows from Fact 4.13 that E^{NP} cannot be computed by $\mathscr{C}[n^k]$ circuits on almost every input length. □

Our equivalence also holds in the high-end regime (e.g., for subexponential-size circuit lower bounds) and in the infinitely-often setting:

Theorem 4.18. *Let $\mathscr{C} \in \{\mathsf{TC}^0, \mathsf{NC}^1, \mathsf{P}_{/\mathrm{poly}}\}$. The following are equivalent:*

1. *There is a constant $\epsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin \mathscr{C}[2^{n^\epsilon}]$.*
2. *There is a constant $\epsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L cannot be $(1/2 + 1/2^{n^\epsilon})$-approximated by $\mathscr{C}[2^{n^\epsilon}]$.*
3. *There is a constant $\epsilon > 0$ and an i.o. $\mathsf{GapUNSAT}$ algorithm for $\mathscr{C}[2^{n^\epsilon}]$ with E^{NP} preprocessing, error $1 - 0.01$, and $2^n/n^{\omega(1)}$ query time.*
4. *There is a constant $k \ge 1$ such that for every good function $s = s(n)$, there is an i.o. CAPP algorithm for $\mathscr{C}[s]$ with $\mathsf{TIME}[2^{\log^k s}]^{\mathsf{NP}}$ preprocessing, $2^{\log^k s}$ query time, and inverse-circuit-size error.*
5. *There is a constant $c \ge 1$ and an E^{NP}-computable i.o. PRG with seed length $\log^c n$ that $(1/n)$-fools $\mathscr{C}$ circuits of size n.*

Proof Sketch. (3) $\implies$ (1) follows from Corollary 4.3.
(1) $\implies$ (2) follows from Theorem 4.15.
(2) $\implies$ (5) follows from Theorem 4.7.
(5) $\implies$ (4) follows by simply applying the i.o. PRG to fool the input circuit.
(4) $\implies$ (3) is trivial. □

For weaker circuit classes, we also get an equivalence by considering *strong average-case* lower bounds: hard functions that cannot be $(1/2 + 1/\mathrm{poly}(n))$-approximated by $\mathscr{C}$, non-trivial CAPP data structures for $\mathscr{C}$ with inverse-circuit-size error, and PRGs fooling $\mathscr{C}$ are equivalent. Note that

the following equivalence only holds in the low-end regime (i.e., for polynomial size but not for subexponential size), and only holds infinitely often. The reason is that we need to use a win-win argument in [CR20, CLLO21]: if a certain NC^1-hard problem (called DCMD in [CR20]) can be approximated by $\mathscr{C}$ circuits, we proceed in one way; if not, we proceed in another way. The reader is referred to the discussion in [CLW20, Section 2.2.2] for more details on the limitation of this win-win argument.

Below we list the properties of the weak circuit class $\mathscr{C}$ that we need:

($\mathscr{C}$ is typical) $\mathscr{C}$ is closed under negation and projection.

($\mathscr{C}$ is universal) For every truth table of length 2^k, there is a $\mathscr{C}$ circuit of size $\mathrm{poly}(2^k)$ that computes this truth table.

($\mathscr{C}$ computes PARITY) The PARITY function can be computed by a $\mathscr{C}$ circuit of size $\mathrm{poly}(n)$.

($\mathscr{C}$ is closed under bottom juntas) For every constant $d \geq 1$, every $\mathscr{C} \circ \mathsf{Junta}_d$ circuit can be computed by a polynomially-larger $\mathscr{C}$ circuit.

($\mathscr{C}$ is closed under top NC^0 circuits) For every constant $d \geq 1$, every $\mathsf{NC}^0_d \circ \mathscr{C}$ circuit can be computed by a polynomially-larger $\mathscr{C}$ circuit.

Theorem 4.19. *Let $\mathscr{C}$ be a circuit class that satisfies the properties above. If $\mathscr{C} \subseteq \mathsf{NC}^1$, then the following are equivalent:*

1. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ that cannot be $\left(1/2 + 1/n^k\right)$-approximated by $\mathscr{C}[n^k]$.*
2. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin \mathsf{MAJ} \circ \mathscr{C}[n^k]$.*
3. *There is $\delta \geq 1/\mathrm{poly}(n)$ such that for every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin \widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}[n^k]$.*
4. *For every constant $k \geq 1$, there is a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that $L \notin \widetilde{\mathsf{Sum}}_{1/3} \circ \mathscr{C}[n^k]$.*
5. *For every constant $k \geq 1$, there is an infinitely-often CAPP algorithm for $\mathscr{C}[n^k]$ with E^{NP} preprocessing, $2^n/n^{\omega(1)}$ query time, and inverse-circuit-size error.*
6. *For every constant $\delta > 0$ and every good function $s = s(n)$, there is an infinitely-often CAPP algorithm for $\mathscr{C}[s]$ circuits with $\mathsf{TIME}[\exp(s^\delta)]^{\mathsf{NP}}$ preprocessing, $\exp(s^\delta)$ query time, and inverse-circuit-size error.*
7. *For every constant $k \geq 1$, there is an E^{NP}-computable i.o. PRG with seed length $n - 1$ that $\left(1/n^k\right)$-fools $\mathscr{C}[n^k]$ circuits.*
8. *For every constant $k \geq 1$, there is an E^{NP}-computable i.o. PRG with seed length $n^{1/k}$ that $\left(1/n^k\right)$-fools $\mathscr{C}[n^k]$ circuits.*

Proof. (8) $\Longrightarrow$ (7), (6) $\Longrightarrow$ (5), and (4) $\Longrightarrow$ (3) are trivial. (2) $\Longrightarrow$ (4) follows from the fact that $\bigcup_{k\in\mathbb{N}} \widetilde{\mathsf{Sum}}_{1/3} \circ \mathscr{C}[n^k] \in \bigcup_{k\in\mathbb{N}} \mathsf{MAJ} \circ \mathscr{C}[n^k]$.

(8) $\Longrightarrow$ (6): The proof is exactly the same as (6) $\Longrightarrow$ (4) in Theorem 4.17. That is, we apply the PRG to solve CAPP.

(7) $\Longrightarrow$ (3): From Lemma 4.10, a PRG fooling $\mathscr{C}$ is also a PRG fooling $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$. Then, from Fact 4.13, a PRG fooling $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$ implies a lower bound for it. Details follow.

For any fixed constant k, let $G_n : \{0,1\}^{n-1} \to \{0,1\}^n$ be an E^{NP}-computable i.o. PRG that $(1/n^{k+1})$-fools $\mathscr{C}[n^{k+1}]$ circuits. Then, by Lemma 4.10, G is also an i.o. PRG that ϵ'-fools $\widetilde{\mathsf{Sum}}_{1/n} \circ \mathscr{C}$ circuits of complexity n^k, where

$$\epsilon' := 2 \cdot (1/n) + n^k/n^{k+1} = 3/n < 2^{(n-1)-n} = 1/2.$$

By Fact 4.13, there is a language in E^{NP} that is not computable in $\widetilde{\mathsf{Sum}}_{1/n} \circ \mathscr{C}[n^k]$.

(5) $\implies$ (3) follows from arguments similar to Corollary 4.3. Let $\epsilon := 1/6$, then by Corollary 4.3, there is a constant d such that the following holds. For any constant k, non-trivial i.o. $\mathsf{GapUNSAT}$ algorithms for $\mathsf{NC}^0_d \circ \widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}[n^k]$ with error ϵ imply the $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}[n^k]$ lower bound we are looking for. Here δ is a constant that we will assign later. We then use Lemma 4.11 to obtain the required i.o. $\mathsf{GapUNSAT}$ algorithm.

Note that we cannot apply Corollary 4.3 directly in the above argument for the following reason. In Corollary 4.3, the nondeterministic machine M^{PCPP} guesses H size-$s(n)$ circuits $C_1, C_2, \ldots, C_H$ and uses the truth tables of these circuits as rows of the proof matrix; then it runs the i.o. $\mathsf{GapUNSAT}$ algorithm to decide whether VPCPP accepts the proof matrix. The problem is that in our case, the circuits are $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuits, which are $\mathsf{Sum} \circ \mathscr{C}$ circuits with the *promise* that its ℓ_∞-distance to the closest Boolean function is less than δ. Thus we need to additionally verify that the $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuits satisfy the promise, as the i.o. $\mathsf{GapUNSAT}$ algorithm might behave incorrectly on arbitrary $\mathsf{Sum} \circ \mathscr{C}$ circuits.

We apply Lemma 4.12 to check whether the guessed $\mathsf{Sum} \circ \mathscr{C}$ circuits are "valid" for our i.o. $\mathsf{GapUNSAT}$ algorithm. For a $\mathsf{Sum} \circ \mathscr{C}$ circuit $C := \sum_{i=1}^{S} \alpha_i \cdot C_i$, Lemma 4.12 presents an algorithm that always accepts when $\|C - \mathsf{bin}_C\|_\infty$ is small and always rejects when $\|C - \mathsf{bin}_C\|_d$ is large.[24]

More precisely, recall that we defined d to be the constant from Corollary 4.3 given $\epsilon = 1/6$. Let δ_{req} be the constant from Lemma 4.11 such that given d $\mathsf{Sum} \circ \mathscr{C}$ circuits $C_1, \ldots, C_d$, if $\|C_i - \mathsf{bin}_{C_i}\|_d \le \delta_{\mathsf{req}}$, we can estimate the following quantity within error $1/6 \cdot 2^{-d}$ in $2^n/n^{\omega(1)}$ time:

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n} [\mathsf{bin}_{C_1}(x) \wedge \mathsf{bin}_{C_2}(x) \wedge \cdots \wedge \mathsf{bin}_{C_d}(x)].$$

Our $\mathsf{GapUNSAT}$ algorithm for $\mathsf{NC}^0_d \circ \mathsf{Sum} \circ \mathscr{C}$ works as follows. Let $C = f(\mathsf{bin}_{C_1}, \mathsf{bin}_{C_2}, \ldots, \mathsf{bin}_{C_d})$ be an $\mathsf{NC}^0_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit, where $f : \{0,1\}^d \to \{0,1\}$ is an arbitrary Boolean function, and each C_i is a $\mathsf{Sum} \circ \mathscr{C}$ satisfying the promise (i.e., $\|C_i - \mathsf{bin}_{C_i}\|_d \le \delta_{\mathsf{req}}$). We applying the above CAPP algorithm for $O(2^d)$ times:

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n} [f(\mathsf{bin}_{C_1}(x), \ldots, \mathsf{bin}_{C_d}(x))] = \sum_{v \in \{0,1\}^d, f(v)=1} \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n} [\mathsf{bin}_{C_1}(x) = v_1 \wedge \cdots \wedge \mathsf{bin}_{C_d}(x) = v_d]$$

The error will be at most $2^d \cdot (1/6 \cdot 2^{-d}) = 1/6$ and will thus satisfy the requirement of Corollary 4.3. It follows that if we can guarantee (by the testing algorithm) that every $\mathsf{Sum} \circ \mathscr{C}$ circuit C_i that we guess satisfies $\|C_i - \mathsf{bin}_{C_i}\|_d \le \delta_{\mathsf{req}}$, then we obtain a lower bound against $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuits.

To test the circuits, we apply Lemma 4.12 with parameter δ_{req}. More precisely, let c be a large enough constant such that $c \ge (2d+1)k$ and any $\mathsf{AND}_{2d} \circ \mathscr{C}[n^k]$ circuit can be simulated by a size-n^c $\mathscr{C}$ circuit. By (5), there is an infinitely-often CAPP algorithm for $\mathscr{C}[n^c]$ with $2^n/n^{\omega(1)}$ query time, inverse-circuit-size error, and E^{NP} preprocessing. As the $\mathsf{Sum} \circ \mathscr{C}$ circuit being tested has

[24] Note that this is not an exact verification algorithm: if the verification accepts, we are only guaranteed that C has small ℓ_d-distance to a Boolean function. But this guarantee is strong enough to ensure our $\mathsf{GapUNSAT}$ algorithm is correct.

complexity $\leq n^k$, we have that $\alpha_{\mathsf{tot}} \leq n^k$ and thus $\frac{\delta_{\mathsf{req}}^d}{2\cdot 3^d\cdot(\alpha_{\mathsf{tot}}+1)^{2d}} \geq n^{-c}$. The CAPP problem for $\mathsf{AND}_{2d} \circ \mathscr{C}$ circuits can be estimated within error n^{-c} in $2^n/n^{\omega(1)}$ time. Thus we have that each $\mathsf{Sum} \circ \mathscr{C}$ circuit can be checked in time $(2^n/n^{\omega(1)}) \cdot \mathrm{poly}(n) \leq 2^n/n^{\omega(1)}$ time.

Therefore, all $\widetilde{\mathsf{Sum}}_{\delta_{\mathsf{req}}/3} \circ \mathscr{C}$ circuits will be accepted by the testing algorithm. Setting $\delta := \delta_{\mathsf{req}}/3$, we have that (3) holds for $\widetilde{\mathsf{Sum}}_{\delta} \circ \mathscr{C}[n^k]$ circuits.

(3) $\implies$ (1) follows from the proof of (2) $\implies$ (6) in [CLLO21, Theorem 1]; for completeness we provide a sketch here. If a certain problem called DCMD, which is in P, cannot be $(1/2+1/n^k)$-approximated by $\mathscr{C}[n^k]$ for every constant k, then (1) follows directly. Otherwise, by [CR20, Lemma 3.1], $\mathsf{NC}^1 \subseteq \widetilde{\mathsf{Sum}}_{\delta} \circ \mathscr{C}$. Let L be a language in E^{NP} that does not have $\widetilde{\mathsf{Sum}}_{\delta} \circ \mathscr{C}[n^{O(k)}]$ circuits, then L does not have NC^1 circuits of size $n^{O(k)}$. By standard hardness amplification (with NC^1 decoders) such as Theorem 4.15, it follows that L cannot be $(1/2 + 1/n^k)$-approximated by NC^1 circuits of size $n^{O(k)}$. Since $\mathscr{C} \subseteq \mathsf{NC}^1$, (1) is true.

(1) $\implies$ (2) follows from the discriminator lemma [HMP+93]. For a function $f \in \mathsf{MAJ} \circ \mathscr{C}$ where the top MAJ gate has fan-in s, f can be $(1/2 + 1/O(s))$ approximated by $\mathscr{C}$. This implies the contrapositive of (1) $\implies$ (2).

(1) $\implies$ (8) follows from the Nisan-Wigderson generator [NW94]. We use the following parameters in Theorem 4.17:

$$\ell := n^{1/3k}, m := n, a := 4k, t := O(\ell^2 \cdot m^{1/a}/a) := O(\ell^2) \leq n^{1/k}.$$

We use a hard truth table of length 2^ℓ that is not $(1/2+1/n^{k+1})$-approximated by $\mathscr{C} \circ \mathsf{Junta}_a$ circuits, where the top $\mathscr{C}$ circuits have size n^k. As any $\mathscr{C} \circ \mathsf{Junta}_a$ can be computed by a polynomially larger $\mathscr{C}$, the same proof applies. □

We also present a characterisation of E^{NP} lower bounds against weak circuit classes that holds both in the high-end regime and almost everywhere. Note that we cannot consider lower bounds against $\widetilde{\mathsf{Sum}} \circ \mathscr{C}$ circuits here; instead, we use $[0,1]\text{-}\mathsf{Sum} \circ \mathscr{C}$ circuits that are close to Boolean in ℓ_1-distance. Also, for technical reasons (that arise when applying Lemma 4.14), we need to consider *seed-extending* PRGs.

Theorem 4.20. *Let $\mathscr{C}$ be a circuit class satisfying the above properties. Moreover, suppose the property that $\mathscr{C}$ is closed under bottom juntas is strengthened to:*

($\mathscr{C}$ is closed under bottom juntas) *Every $\mathscr{C} \circ \mathsf{Junta}_{\log n}$ circuit can be computed by a polynomially large $\mathscr{C}$ circuit.*

Then the following are equivalent:

1. *There is a constant $\epsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L cannot be $(1/2 + 1/2^{n^\epsilon})$-approximated by $\mathscr{C}[2^{n^\epsilon}]$ circuits on almost every input length.*
2. *There is a constant $\epsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L cannot be approximated by $[0,1]\text{-}\mathsf{Sum} \circ \mathscr{C}[2^{n^\epsilon}]$ circuits within ℓ_1-distance $1/3$ on almost every input length.*
3. *There is a constant $\epsilon > 0$ and a CAPP algorithm for $\mathscr{C}[2^{n^\epsilon}]$ with E^{NP} preprocessing, 2^{n-n^ϵ} query time, and inverse-circuit-size error.*
4. *There is a constant $c \geq 1$ such that for every good function $s(n)$, there is a CAPP algorithm for $\mathscr{C}[s]$ circuits with E^{NP} preprocessing, $2^{\log^c s}$ query time, and inverse-circuit-size error.*

5. *There is a constant $\epsilon > 0$ and a seed-extending E^{NP}-computable PRG with seed length $n-1$ that $(1/2^{n^\epsilon})$-fools $\mathscr{C}$ circuits of size 2^{n^ϵ}.*

6. *There is a constant $c \geq 1$ and a seed-extending E^{NP}-computable PRG with seed length $\log^c n$ that $(1/n)$-fools $\mathscr{C}$ circuits of size n.*

Proof. (4) $\implies$ (3) is trivial.

(6) $\implies$ (5) can be proved by padding the circuit with dummy inputs.

(6) $\implies$ (4): The proof is the same as (6) $\implies$ (4) in Theorem 4.17.

(5) $\implies$ (2) follows from Lemma 4.14.

(3) $\implies$ (2) follows from careful adaption of Theorem 4.2, which is similar to the proof of (5) $\implies$ (3) in Theorem 4.19. The only difference is the verification algorithm: given a $\mathsf{Sum} \circ \mathscr{C}$ circuit C as input, we need to accept when C is a $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit where $\|C - \mathsf{bin}_C\|_1$ is small (instead of when $\|C - \mathsf{bin}_C\|_\infty$ is small), and reject when $\|C - \mathsf{bin}_C\|_d$ is large. Therefore, we need to use the second (instead of the first) condition in Lemma 4.12. That is, we need to set $\delta := (\delta_{\mathsf{req}}/3)^d$ instead of $\delta_{\mathsf{req}}/3$, so that if $\|C - \mathsf{bin}_C\|_1 \leq \delta$ then the algorithm accepts. This different will not introduce new problems as δ is still a constant. After the verification algorithm accepts, we know that every $\mathsf{Sum} \circ \mathscr{C}$ circuit we guess has ℓ_d-distance at most δ_{req} to Boolean, and our $\mathsf{GapUNSAT}$ algorithm is correct.

(2) $\implies$ (1) follows from the XOR lemma in [CLW20]. In particular, suppose that $L \in \mathsf{E}^{\mathsf{NP}}$ cannot be approximated by $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}[2^{n^\epsilon}]$ circuits within ℓ_1-distance $1/3$ on almost every input length. Let $\delta := 1/3$, $k := O(n^{\epsilon/2})$, and

$$\epsilon_k := (1-\delta)^{k-1}(1/2 - \delta) < 2^{-n^{\epsilon/2}}.$$

By Theorem 4.16, $L^{\oplus k}$ cannot be $(1/2+\epsilon_k)$-approximated by $\mathscr{C}$ circuits of size $2^{n^{\epsilon/2}}$. Since $L \in \mathsf{E}^{\mathsf{NP}}$, we also have $L^{\oplus k} \in \mathsf{E}^{\mathsf{NP}}$. Note that $L^{\oplus k}$ is a function on $\ell := kn = O(n^{1+\epsilon/2})$ input bits. Therefore, for $\epsilon' := 0.49\epsilon/(1+\epsilon)$, $L^{\oplus k}$ cannot be $(1/2 + 1/2^{\ell^{\epsilon'}})$-approximated by $\mathscr{C}$ circuits of size $2^{\ell^{\epsilon'}}$.

(1) $\implies$ (6): Let $L \in \mathsf{E}^{\mathsf{NP}}$ be a language that cannot be $(1/2 + 1/2^{n^\epsilon})$-approximated by $\mathscr{C}[2^{n^\epsilon}]$ circuits on almost every input length. Let $c \geq 2$ be a large enough constant such that $\mathscr{C} \circ \mathsf{Junta}_a$ circuits where the top $\mathscr{C}$ circuit has size n can be simulated by $\mathscr{C}$ circuits of size n^c. We apply the Nisan-Wigderson generator [NW94] with the following parameters:

$$\ell := \log^{c/\epsilon} n, m := n, a := \log n, t := O(\ell^2 \cdot m^{1/a}/a) := O(\ell^2) \leq \log^{2c/\epsilon} n.$$

From (1) we have that there exists a function $f : \{0,1\}^\ell \to \{0,1\}$ in E^{NP} that cannot be $(1/2+1/2^{\ell^\epsilon}) = (1/2+1/n^c)$-approximated by $\mathscr{C}$ circuits of size n^c. Then f cannot be $(1/2+1/n^c)$-approximated by $\mathscr{C}[n] \circ \mathsf{Junta}_a$ circuits. By Theorem 4.7, there is a function $G : \{0,1\}^{2^\ell} \times \{0,1\}^t \to \{0,1\}^m$ computable in $\mathrm{poly}(m, 2^t) \leq 2^{O(t)}$ time, such that $G(tt(f), -)$ is a PRG that $(1/n^{c-1})$-fools every $\mathscr{C}[n]$ circuit. Since $f \in \mathsf{E}^{\mathsf{NP}}$, the generator $G(tt(f), \cdot)$ is computable in E^{NP}. □

5 On the Limits of (Unconditional) Range Avoidance Algorithms

In this section, we investigate the role of stretch functions in the complexity of Avoid. We show that if the stretch function is small enough, then range avoidance algorithms (even in $\mathsf{FP}^{\mathsf{NP}}$) imply breakthrough lower bounds, indicating that such algorithms may be beyond reach.

5.1 Breakthrough Lower Bounds from AC^0-Avoid

In this sub-section, we show that solving AC^0-Avoid with polynomial stretch (actually, "fixed-quasi-polynomial" stretch) implies breakthrough circuit lower bounds such as $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{NC}^1$.

Definition 5.1. Let $\mathscr{C}$ be a circuit class. We say that $\mathscr{C}$ has the *universality property* if there is a constant $c \geq 1$ such that for any good function $s : \mathbb{N} \to \mathbb{N}$, there is a sequence of $\mathscr{C}$-circuits $\{U_{s,n}\}_{n\in\mathbb{N}}$ such that the following are true:

- The size of $U_{s,n}$ is $s(n)^c$ and it has $O(s \log s + n)$ variables.
- Given an input $(\langle C\rangle, x)$, where $\langle C\rangle$ is the encoding of a $\mathscr{C}$-circuit C of size s on n variables, and $x \in \{0,1\}^n$, it accepts the input iff C accepts x.
- The family $U_{s,n}$ is uniform: there is a Turing machine that on input $(1^s, 1^n)$, outputs the description of $U_{s,n}$ in polynomial time.

Roughly speaking, the above definition means that $\mathscr{C}$ has the *universal property* if the $\mathscr{C}$-Eval problem (given the description of a $\mathscr{C}$ circuit C and an input x, compute $C(x)$) is itself computable in $\mathscr{C}$. The following theorem shows that for any circuit class $\mathscr{C}$ with the universal property, solving $\mathscr{C}$-Avoid implies lower bounds against $\mathscr{C}$.

Theorem 5.2. *Let $\mathscr{C}$ be any circuit class that has the universality property, and $f : \mathbb{N} \to \mathbb{N}$ be a monotone function that is good. Suppose there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid, where the $\mathscr{C}$ circuits have input length N, output length $f(N)$, and each output gate has $\mathscr{C}$ circuit complexity* $\mathrm{poly}(N)$. *Then for some constant $\epsilon > 0$, E^{NP} does not have $\mathscr{C}$-circuits of size $f^{-1}(2^n)^\epsilon$.*

Proof. Let c be the constant corresponding to the universality property of $\mathscr{C}$, without loss of generality assume $c > 1$. Let $\epsilon := 1/c$ and $s(n) := f^{-1}(2^n)^\epsilon$. Let $\{U_{s,n}\}$ be the sequence of universal $\mathscr{C}$-circuits corresponding to size parameter $s(n)$, then the size of $U_{s,n}$ is $s(n)^c$. We show how to use the universality of $\mathscr{C}$ together with the avoidance algorithm to derive $\mathscr{C}$-lower bounds for E^{NP}.

Consider the *truth table* map

$$\mathsf{TT}_{\mathscr{C}} : \{0,1\}^N \to \{0,1\}^{2^n}$$

defined as follows. The input of $\mathsf{TT}_{\mathscr{C}}$ is the encoding $\langle C\rangle$ of a (single-output) $\mathscr{C}$ circuit of size $s(n)$ on n variables. (Thus the input length is $N := O(s \log s)$.) The output is the truth table of C, i.e. the string of length 2^n which specifies $C(x)$ for each $x \in \{0,1\}^n$ in lexicographic order. By the universality of $\mathscr{C}$, this output can be computed by running $U_{s,n}$ on the input $(\langle C\rangle, x)$ for each input $x \in \{0,1\}^n$. Hence, $\mathsf{TT}_{\mathscr{C}}$ can be implemented by a multi-output $\mathscr{C}$-circuit with

$$2^n = f(s(n)^c) \geq f(N)$$

output bits, where each output bit is computable by a $\mathscr{C}$-circuit of size $s(n)^c \leq \mathrm{poly}(N)$.

Now we apply the $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid and obtain the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ that is not in $\mathrm{Range}(\mathsf{TT}_{\mathscr{C}})$. Note that $\mathrm{Range}(\mathsf{TT}_{\mathscr{C}})$ is exactly the set of truth tables computable by $\mathscr{C}$ circuits of size $s(n)$, thus f cannot be computed by size-$s(n)$ $\mathscr{C}$ circuits. Note that f is in E^{NP} because its truth table can be obtained in P^{NP} as a function of 2^n (which is the output length of $\mathsf{TT}_{\mathscr{C}}$), and hence in E^{NP} as a function of n. □

Theorem 5.2 can be used to show that even for *weak* classes such as AC^0 and *super-polynomial* stretch, the range avoidance problem might be beyond reach — since a deterministic algorithm for it implies breakthrough circuit lower bounds.

Lemma 5.3 ([CH85]). *The circuit class* $\mathsf{AC}^0 = \bigcup_d \mathsf{AC}^0_d$ *has the universality property.*

Corollary 5.4. *Suppose that there is a constant* $b \geq 1$ *such that for every depth constant* $d \geq 1$*, there is an* $\mathsf{FP}^{\mathsf{NP}}$ *algorithm for* AC^0_d-Avoid *with input length* N *and output length* $2^{\log^b N}$*. Then* E^{NP} *is not contained in* NC^1*.*

Proof. First, by Lemma 5.3, we can apply Theorem 5.2 to the circuit class AC^0. A direct application of Theorem 5.2 yields that E^{NP} does not have AC^0_d circuits of size 2^{n^δ} for some fixed $\delta > 0$ and any depth d. Now we use the well-known simulation result [Nep70, AHM+08]: for *every* $\delta > 0$, there is some depth $d \geq 1$ such that NC^1 is contained in $\mathsf{AC}^0_d[2^{n^\delta}]$. Combining this simulation result with the lower bound arising from Theorem 5.2 yields that E^{NP} is not contained in NC^1. □

5.2 Breakthrough Lower Bounds from NC^0-Avoid

Next, we proceed to show that even for the simplest non-trivial class of circuits, i.e., NC^0 (each output bit only depends on a *constant* number of input bits), $\mathsf{FP}^{\mathsf{NP}}$ avoidance algorithms in the small-stretch regime would imply breakthrough lower bounds. To establish this, we use the *randomised encoding* technique of [AIK06]. Roughly speaking, NC^1-Avoid can be reduced to NC^0-Avoid (in the small-stretch regime) using the randomised encoding technique, with any solution to the latter problem yielding a solution to the former problem.

We state the randomised encoding property in a way that is convenient for our applications and is implied by the standard definitions [AIK06].[25]

Definition 5.5. Let $\ell = \ell(n), m = m(n)$ be good functions, and consider functions

$$f_n : \{0,1\}^n \to \{0,1\}^\ell \quad \text{and} \quad \hat{f}_n : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^{\ell+m}.$$

We say that $\hat{f}$ is a *perfect randomised encoding* of f if there is a a polynomial-time computable decoder $\mathsf{Dec} : \{0,1\}^{\ell+m} \to \{0,1\}^\ell$ such that for every $x \in \{0,1\}^n$ and $y \in \{0,1\}^{\ell+m}$, $f(x) = \mathsf{Dec}(y)$ iff there is $r \in \{0,1\}^m$ such that $\hat{f}(x,r) = y$.

Intuitively, the definition means that from the value of $\hat{f}(x,r)$ for *any* r, we can decode $f(x)$ without knowing x. Moreover, for every $x \in \{0,1\}^n$ and $y \in \{0,1\}^{\ell+m}$ such that y is decoded to $f(x)$, there is some (actually, a unique) r such that $\hat{f}(x,r) = y$. Therefore, the range of $\hat{f}(x,-)$ is indeed an "encoding" of the value $f(x)$.

This definition only makes sense if $\hat{f}$ is much easier to compute that f. This is actually the case: [AIK06] showed that even very powerful functions f (computable by polynomial-size formulas) have perfect randomised encodings $\hat{f}$ computable in NC^0_4. That is, every output bit of $\hat{f}$ only depends on *four* input bits.

Lemma 5.6 ([AIK06]). *Let* $\ell = \ell(n)$ *and* $s = s(n)$ *be good functions. Every sequence of multi-output functions* $\{f_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\}$ *computable by size-*$s(n)$ *formulas has a perfect randomized encoding* $\hat{f} : \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}^{\ell(n)+m(n)}$ *computable in* NC^0_4*, for which* $m(n) \leq \mathrm{poly}(n, \ell(n), s(n))$*.*

Lemma 5.7 ([Bus87]). *The class of formulas has the universality property.*

Now we prove the main theorem in this sub-section: solving the range avoidance problem for NC^0_4, which is a very "simple" class, implies breakthrough lower bounds.

[25] See discussions in "A combinatorial view of perfect encoding" on Page 9 of [AIK06].

Theorem 5.8. *Suppose that for every constant $\epsilon > 0$ there is an $\mathsf{FP}^{\mathsf{NP}}$ (resp. FP) algorithm for NC^0_4-Avoid with input length N and output length $N + N^\epsilon$. Then $\mathsf{E}^{\mathsf{NP}} \not\subseteq \mathsf{Formula}[2^{o(n)}]$ (resp. $\mathsf{E} \not\subseteq \mathsf{Formula}[2^{o(n)}]$).*

Proof. We first show, using the randomised encoding technique, that the assumed algorithm for NC^0_4-Avoid implies an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for Formula-Avoid with input length n and output length $\ell := n^{1+\gamma}$ for every constant $\gamma > 0$. We then use Lemma 5.7 and Theorem 5.2 to conclude the formula lower bound.

Let $f : \{0,1\}^n \to \{0,1\}^\ell$ be the input of the range avoidance problem where each output gate of f can be computed by a formula of size $s = \mathrm{poly}(n)$. Let $m := \mathrm{poly}(n, \ell, s) \le \mathrm{poly}(n)$, we consider the function $\hat{f} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^{\ell+m}$, where $\hat{f}$ is the randomised encoding of f defined in Lemma 5.6, and is hence implementable in NC^0_4. Since

$$(\ell + m) - (n + m) = \ell - n \ge (n + m)^{\Omega(1)},$$

by our hypothesis, the range avoidance problem $\hat{f}$ can be solved in $\mathsf{FP}^{\mathsf{NP}}$.

We show how to use this to solve the original range avoidance problem in $\mathsf{FP}^{\mathsf{NP}}$. Let M be an $\mathsf{FP}^{\mathsf{NP}}$ algorithm producing a non-output y for $\hat{f}$. We define an $\mathsf{FP}^{\mathsf{NP}}$ algorithm M' producing a non-output z of f as follows: M' simply produces $z = \mathsf{Dec}(y)$.

Note that since Dec is polynomial-time computable and M is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm, we have that M' is also an $\mathsf{FP}^{\mathsf{NP}}$ algorithm. Next we argue that if y is a non-output of $\hat{f}$, then $\mathsf{Dec}(y)$ is a non-output of f. Indeed, by Definition 5.5, if $\mathsf{Dec}(y) = f(x)$ for some x, then there is $r \in \{0,1\}^m$ such that $y = \hat{f}(x, r)$, i.e., y is an output of $\hat{f}$, which is a contradiction.

Finally, we use the algorithm for Formula-Avoid together with the universality of formulas (Lemma 5.7) to obtain lower bounds. By Theorem 5.2 with f a large enough polynomial, we get that E^{NP} does not have subexponential-size formulas.

In order to see that we can get lower bounds in E from avoidance algorithms in FP, simply note that the reduction from Formula-Avoid to NC^0-Avoid, as well as the arguments in Theorem 5.2, do not use the NP oracle. □

Remark 5.9. The only properties of formulas that are used in the above proof are the universality property of formulas (Lemma 5.7) and the randomised encodings for formulas (Lemma 5.6).

It is shown in [AIK06] that branching programs and parity branching programs also have randomised encodings computable in NC^0_4. Since the (parity) branching program evaluation problem can be easily done in (parity) log-space, the universality of (parity) branching programs is easy to see. Thus, the hypothesis in Theorem 5.8 actually implies the stronger lower bounds that E^{NP} (or E) does not have sub-exponential-size (parity) branching programs.

Discussion. The above proof shows that in the regime with very small stretch (e.g., $\ell(n) = n+1$), Formula-Avoid reduces to NC^0_4-Avoid in polynomial time. We think this result reveals some fundamental difference between the small-stretch regime ($\ell(n) = n + 1$), for which an avoidance algorithm for NC^0 implies breakthrough lower bounds, and the large-stretch regime ($\ell(n) = n^{1+\Omega(1)}$), for which an avoidance algorithm for NC^0 seems within reach (Theorem 3.12).

6 The FNP and FP Regimes

In this section, we present some preliminary results on the complexity of Avoid with respect to algorithms less powerful than $\mathsf{FP}^{\mathsf{NP}}$, such as FNP algorithms and FP algorithms.

6.1 Preliminaries

6.1.1 Proof Complexity

We introduce basic notions in proof complexity. Let TAUT denote the set of DNF tautologies, then TAUT is coNP-complete. A propositional proof system (PPS) is essentially a nondeterministic algorithm trying to solve TAUT:

Definition 6.1 ([CR79])**.** A *propositional proof system* (PPS) is a polynomial-time computable binary relation $Q \subseteq \{0,1\}^\star \times \{0,1\}^\star$ such that:

- For every $\phi, \pi \in \{0,1\}^\star$, if $(\phi, \pi) \in Q$, then $\phi \in \mathsf{TAUT}$.
- For every $\phi \in \mathsf{TAUT}$, there is a string $\pi \in \{0,1\}^\star$ such that $(\phi, \pi) \in Q$.

Here, ϕ is the tautology to be proved and π is the proof.[26]

We define proof complexity generators:

Definition 6.2 ([ABRW04])**.** Let $s(n) < n$ be a function for seed length. A proof complexity generator is a map $C_n : \{0,1\}^s \to \{0,1\}^n$ computed by a family of polynomial-size circuits $\{C_n\}_n$. A generator is *secure* against a propositional proof system P if for every large enough n and every $y \in \{0,1\}^n$, P does not have a polynomial-size proof of the (properly encoded) statement

$$\forall x \in \{0,1\}^s, C_n(x) \neq y.$$

We also consider uniform and non-uniform proof complexity generators. A *uniform* proof complexity generator is a sequence of generators $C_n : \{0,1\}^s \to \{0,1\}^n$ such that there is a polynomial-time Turing machine that on input 1^n generates the circuit C_n. A *non-uniform* proof complexity generator is an arbitrary sequence of generators $C_n : \{0,1\}^s \to \{0,1\}^n$.

6.1.2 Time-Bounded Kolmogorov Complexity

In this section, we define time-bounded Kolmogorov complexity and related notions.

We need to fix a universal Turing machine U in the following definition. In particular, given a program d, an input x, and a time bound t, $U(d, x, 1^t)$ executes d on input x for t steps, and outputs the output of d. Any machine U that runs in $\mathrm{poly}(|d|, |x|, t)$ steps suffices for our purpose.

Definition 6.3. Let $t = t(n)$ be a polynomial and x, y be strings.

- The t-time bounded Kolmogorov complexity of x, denoted as $\mathrm{K}^t(x)$, is the length of the shortest description d such that $U(d, n, 1^{t(n)}) = x$, where $n = |x|$.
- The t-time bounded Kolmogorov complexity of x conditioned on y, denoted as $\mathrm{cK}^t(x \mid y)$, is the length of the shortest description d such that $U(d, y, 1^{t(|d|+|y|)}) = x$.

Note that in the definition of $\mathrm{K}^t(x)$, we assume that $n = |x|$ is already known to the program that generates x. This is for technical convenience.

We define the following explicit construction problems where we want to construct strings with high time-bounded Kolmogorov complexity.

[26] Note that the length of π could be super-polynomially larger than the length of ϕ. In fact, there is a polynomial-bounded PPS (i.e., any tautology ϕ has a polynomially-long proof π) if and only if NP = coNP.

Definition 6.4. Let $t = t(n)$ be a polynomial, $\alpha = \alpha(n)$ be a good function such that $\alpha(n) < n$.

- $\mathrm{cK}^t_{n-\alpha(n)}$-Hard is the following problem: Given as inputs 1^n and a string y, find a string $x \in \{0,1\}^n$ such that $\mathrm{cK}^t(x \mid y) \geq n - \alpha(n)$.
- $\mathrm{K}^t_{n-\alpha(n)}$-Hard is the following problem: Given input 1^n, find a string $x \in \{0,1\}^n$ such that $\mathrm{K}^t(x) \geq n - \alpha(n)$.

We also define a proof complexity generator based on time-bounded Kolmogorov complexity.

Definition 6.5. Fix a function $\alpha = \alpha(n)$, the $\mathrm{K}^t_{n-\alpha(n)}$ generator is the following candidate proof complexity generator $\mathrm{K}^t_{n-\alpha(n)} : \{0,1\}^{n-\alpha(n)} \to \{0,1\}^n$. Given as input the description d of a Turing machine, which has length $n - \alpha(n)$, the generator outputs $U(d, n, 1^{t(n)})$ (i.e., the output of d in $t(n)$ steps), padded/truncated to length n if not already.

6.2 On Avoidance Algorithms in FNP

In this subsection, we characterise the existence of FNP algorithms for Avoid by the existence of propositional proof systems (PPSs) that can break any proof complexity generator. Along the way we also show that the generator $\mathrm{K}^{\mathrm{poly}}_{n-\omega(1)}$ is "complete": modulo some constant loss in the stretch, any PPS fooled by *some* generator is also fooled by $\mathrm{K}^{\mathrm{poly}}_{n-\omega(1)}$.

6.2.1 Non-uniform Proof Complexity Generators vs APEPP

Here we show equivalence results for the case of non-uniform proof generators.

Theorem 6.6. *The following are equivalent:*

1. *There exists a constant c such that* Avoid *is in* FNP *for circuits with stretch c.*
2. *There exists a constant c such that cK^t_{n-c}-Hard is solvable in* FNP *for all polynomials t.*
3. *There exist a constant c and some polynomial $t > 1.1n$ such that cK^t_{n-c}-Hard is solvable in* FNP*.*
4. *There exist a constant c and a propositional proof system P which breaks every non-uniform proof complexity generator of stretch c.*

Proof. (1) $\implies$ (2): Let $(1^n, y)$ be an input of cK^t_{n-c}-Hard, we reduce it to an instance of Avoid. We construct a circuit Φ_y with $n-c$ input bits and n output bits as follows. On input $d \in \{0,1\}^{n-c}$, Φ_y simulates U on input (d, y) for t steps and outputs the result, padded/truncated to length n if not already. Then every non-output of Φ_y will be a valid answer for cK^t_{n-c}-Hard on the input $(1^n, y)$. This is because for a string x that is not a valid answer, there must exist a string d such that $U(d, y)$ outputs x in t steps, thus $\Phi_y(d) = x$ and $x \in \mathrm{Range}(\Phi_y)$.

(2) $\implies$ (3) is trivial.

(3) $\implies$ (4): Let $t > 1.1n$ be the time bound for which we can solve cK^t_{n-c}. Let Eval be the Turing machine that on input a string x and the description $\langle C \rangle$ of a circuit C, outputs $C(x)$. We require the circuit description $\langle C \rangle$ to be properly padded such that Eval runs in time $t(|x| + |\langle C \rangle|)$. The propositional proof system P works as follows: To break a candidate proof complexity generator $C : \{0,1\}^n \to \{0,1\}^m$, we use the proof given to P as the nondeterministic guesses of the FNP

algorithm for cK^t_{n-c}-Hard to get a string y such that $\mathrm{cK}^t(y \mid \langle C \rangle) > m - c$. Now we claim that if $m > n + |d_{\mathsf{Eval}}| + c$ then y would be out of the range of C, where d_{Eval} is the description of the Turing machine Eval. This is because if $y \in \mathrm{Range}(C)$, namely that $y = C(x)$, then as U would output y on input $(d_{\mathsf{Eval}}, x, \langle C \rangle)$ in time $t(n + |\langle C \rangle|)$, we have that $\mathrm{cK}^t(y \mid \langle C \rangle) \leq n + |d_{\mathsf{Eval}}|$.

(4) $\implies$ (1): Given a circuit C as an instance of Avoid, we can simply regard C as a proof complexity generator and use P to break it. To do so, we guess a string x such that $x \notin \mathrm{Range}(C)$ along with a proof π such that $(\phi_x, y) \in P$, where ϕ_x is the DNF encoding of the assertion that $x \notin \mathrm{Range}(C)$. □

6.2.2 Uniform Proof Complexity Generators vs SAPEPP

Here we show equivalence results for the case of uniform proof generators.

The class SAPEPP (for "*sparse* APEPP") is the class of *unary* problems that are reducible to Avoid. More precisely, let $s : \mathbb{N} \to \mathbb{N}$ be a function such that for every integer n, $s(n) < n$. A search problem in SAPEPP_s is defined by a Turing machine M that on input 1^n, outputs in $\mathrm{poly}(n)$ time a circuit $C_n : \{0,1\}^{s(n)} \to \{0,1\}^n$. The total search problem associated with M asks: given n in unary, find a bit-string $y \in \{0,1\}^n$ such that for every $x \in \{0,1\}^{s(n)}$, $M(x) \neq y$.

Now we prove the following theorem. We remark that although the stretch of the generators we consider are $\Theta(\log n)$, there is nothing special with the function $\log n$. It can be replaced by any super-constant function that is good.

Theorem 6.7. *Let $t(n) > 1.1n$ be a polynomial. The following are equivalent:*

1. *For every constant $c > 0$, $\mathrm{K}^t_{n - c\log n}$-Hard is in FNP.*
2. *For every constant $c > 0$, $\mathsf{SAPEPP}_{n-c\log n} \subseteq \mathsf{FNP}$.*
3. *For every constant $c > 0$, there is a propositional proof system P which breaks the $\mathrm{K}^t_{n-c\log n}$ generator.*
4. *For every constant $c > 0$, there is a propositional proof system P which breaks every uniform proof complexity generator with seed length $n - c\log n$.*

Proof Sketch. (3) $\implies$ (1): Let P be the propositional proof system that can break the $\mathrm{K}^t_{n-c\log n}$ generator, then the following FNP algorithm solves $\mathrm{K}^t_{n-c\log n}$-Hard. For $x \in \{0,1\}^n$, let ϕ_x be the DNF encoding of the statement that $\mathrm{K}^t(x) \geq n - c\log n$. We guess a string $x \in \{0,1\}^n$ along with a proof y such that $(\phi_x, y) \in P$. Finally, we output x.

(1) $\implies$ (3): Suppose that $\mathrm{K}^t_{n-c\log n}$ is in FNP, we construct a propositional proof system P as follows. The PPS receives an assertion ϕ and a proof π. If ϕ is of the form ϕ_x for some $x \in \{0,1\}^n$ (recall that ϕ_x is the DNF encoding of the statement that $\mathrm{K}^t(x) \geq n - c\log n$), then we use π as the nondeterministic guesses to the FNP algorithm for $\mathrm{K}^t_{n-c\log n}$-Hard, and obtain a string $y \in \{0,1\}^n$ such that $\mathrm{K}^t(y) \geq n - c\log n$. We then verify that $y = x$. If $y = x$, then we accept (ϕ, π). If $y \neq x$ or the tautology is not of the form ϕ_x, then we use the trivial (e.g., truth-table) proof for ϕ.

The proofs of (2) $\iff$ (4) is similar to (1) $\iff$ (3), so we omit it here.

(2) $\implies$ (1) is trivial.

(1) $\implies$ (2): Let L be a problem in $\mathsf{SAPEPP}_{n-c\log n}$. Then L is defined by a Turing machine M that on input 1^n, outputs a circuit $C_n : \{0,1\}^{n-c\log n} \to \{0,1\}^n$. Let $|d_M|$ be the description length of M, then $|d_M| = O(1)$. For every string $y \in \mathrm{Range}(C_n)$, we have

$$\mathrm{K}^{t'}(y) \leq n - c\log n + |d_M| + O(1) < n - (c/2)\log n$$

for large enough n, where t' is a polynomial that depends on the time complexity of M. Therefore, it suffices to find a string of length n with near-maximum $\mathrm{K}^{t'}$ complexity.

If $t'(n) \leq t(n)$, then we can simply use the FNP algorithm for $\mathrm{K}^t_{n-(c/2)\log n}$-Hard to find the desired string. Now suppose $t'(n) > t(n)$ (which is the more interesting case). Let k be the smallest integer such that $t'(n) < t(n)^{k-1}$ for large enough n,

$$c' := c/(4k) \quad \text{and} \quad n_1 := O(n^k), \tag{3}$$

then

$$O(t'(n)) \leq t(n_1) \quad \text{and} \quad c' \log n_1 \leq (c/3) \log n.$$

We use the FNP algorithm for $\mathrm{K}^t_{n-c'\log n}$-Hard to find a string $y \in \{0,1\}^{n_1}$ such that $\mathrm{K}^t(y) > n_1 - c' \log n_1 = n_1 - (c/3)\log n$. Let y_0 be the first n bits of y, we claim that $\mathrm{K}^{t'}(y_0) \geq n - (c/2)\log n$, which implies the correctness of our algorithm for L.

To see this claim holds, suppose for contradiction that $\mathrm{K}^{t'}(y_0) < n - (c/2)\log n$. Let M_{pad} be the following Turing machine:

> First, given n_1, M_{pad} infers n from Eq. (3). Then, it receives as input a description d of length $n - (c/2)\log n$ and a string $y_1 \in \{0,1\}^{n'-n}$. Let U be the universal Turing machine, M_{pad} runs $U(d)$ for $t'(n)$ steps to obtain a string y_0, padded/truncated to length n if not already. Then it outputs the concatenation of y_0 and y_1.

If d is a description of length $n - (c/2)\log n$ such that $U(d)$ outputs y_0 in $t'(n)$ steps, and y_1 is the last $n' - n$ bits of y, then $M_{\mathsf{pad}}(U, y_1)$ outputs y in $O(t'(n)) \leq t(n_1)$ steps. Let $|d_{M_{\mathsf{pad}}}|$ be the description length of M_{pad}, then we have $\mathrm{K}^{t'}(y) \leq n_1 - (c/2)\log n + |d_{M_{\mathsf{pad}}}| < n_1 - (c/3)\log n$, which is a contradiction. Thus the claim follows. □

In the super-constant-stretch regime, our results have the following implication for proof complexity generators:

Corollary 6.8 (Informal)**.** *There exists a uniform proof complexity generator that is the "hardest", namely the $\mathrm{K}^{1.1n}_{n-\omega(1)}$ generator, in the following sense. If there is a PPS breaking this generator, then there is a PPS breaking any uniform proof complexity generator of seed length $n - \omega(1)$.*

Corollary 6.9 (Informal)**.** *If every uniform proof complexity generator of stretch $n - \omega(1)$ can be broken by some PPS, then there exists a single PPS that breaks every uniform proof complexity generator of stretch $n - \omega(1)$.*

6.3 On Avoidance Algorithms in FP

Finally, we prove equivalence results for $\mathsf{SAPEPP} \subseteq \mathsf{FP}$. Again, in the following theorem, there is nothing special about the function $\log n$; it can be replaced by any super-constant function that is good.

Theorem 6.10. *For any polynomial $t(n) > n^3$, the following are equivalent:*

1. *For every constant $c > 0$, $\mathrm{K}^t_{n-c\log n}$-Hard is in FP.*
2. *For every constant $c > 0$, $\mathsf{SAPEPP}_{n-c\log n} \subseteq \mathsf{FP}$.*
3. *For every constant $c > 0$, there is a language L in $\mathsf{E}\backslash\text{i.o.-}\mathsf{DTIME}[2^{n+1}]_{/(2^n - cn)}$.*

Proof. (2) $\implies$ (1) is trivial.

(1) $\implies$ (3): Suppose we want to find a language in $\mathsf{E} \setminus \text{i.o.-}\mathsf{DTIME}[2^{n+1}]_{/(2^n - cn)}$. Let $c' := c/2$, M be the machine that solves $\mathrm{K}^t_{n-c' \log n}$-HARD, and $tt_n := M(1^{2^n})$. Let M^{hard} be the machine whose truth table is exactly tt_n on input length n. That is, on input $x \in \{0,1\}^n$, M^{hard} outputs the x-th bit of $M(1^{2^n})$. Let L^{hard} be the language accepted by M^{hard}, it is easy to see that $L^{\mathsf{hard}} \in \mathsf{E}$. We claim that $L^{\mathsf{hard}} \notin \text{i.o.-}\mathsf{DTIME}[2^{n+1}]_{/(2^n - cn)}$.

Suppose for contradiction that $L^{\mathsf{hard}} \in \text{i.o.-}\mathsf{DTIME}[2^{n+1}]_{/(2^n - cn)}$. Since M solves $\mathrm{K}^t_{n-c' \log n}$-HARD, we have that $\mathrm{K}^{t(2^n)}(y) > 2^n - c'n$. However, since $L \in \text{i.o.-}\mathsf{DTIME}[2^{n+1}]_{/(2^n - cn)}$, there is a program d of length $\leq 2^n - cn + O(1) < 2^n - c'n$ and time complexity at most 4^{n+1} that on infinitely many n, prints the truth table of L^{hard} on input length n. This contradicts the fact that $\mathrm{K}^{t(2^n)}(tt_n) > 2^n - c'n$.

(3) $\implies$ (2): Let $L \in \mathsf{SAPEPP}_{n - c \log n}$, we want to solve L in polynomial time. Let $\{C_n : \{0,1\}^{n - c \log n} \to \{0,1\}^n\}$ be a uniform family of circuits such that the input 1^n of L is reduced to the circuit C_n. We may pad some dummy inputs and outputs to C_n so that C_n becomes a circuit from $N - 2c' \log N$ bits to N bits where N is a power of 2, and C_n can be evaluated in time $2N$. Here, c' is some constant depending on c and L. Since $\{C_n\}$ is a uniform family, every output of C_n has K^{2N}-complexity at most $N - 2c' \log N + O(1)$.

Now, let $L^{\mathsf{hard}} \in \mathsf{E} \setminus \text{i.o.-}\mathsf{DTIME}[2^{n+1}]_{/(2^n - c'n)}$. Let tt be the truth table of L^{hard} on input length $\log N$. If $\mathrm{K}^{2N}(tt) \leq N - c' \log N - O(1)$ for infinitely many N, then L is in $\text{i.o.-}\mathsf{DTIME}[2^{n+1}]_{/(2^n - c'n)}$, contradicting (3). Therefore $\mathrm{K}^{2N}(tt) > N - c' \log N - O(1) > N - 2c' \log N$. It follows that tt is a non-output of C_n. Since $L^{\mathsf{hard}} \in \mathsf{E}$, we can compute tt in polynomial time, thus solving L. □

7 A Rectangular PCP of Proximity

In this section, we prove Theorem 2.9 by building a rectangular PCP of proximity.

Organisation of this section. Our rectangular PCPP is based on the PCPP in [BGH+06, BGH+05], which we review in Section 7.1.1. As our main goal is to verify rectangularity, we will focus on the *query pattern* of the verifier. In Section 7.1.2, we show the verifier is indeed almost rectangular. It is shown in [BHPT20] that the PCP verifier is rectangular; but to deal with the PCP *of proximity* verifier, additional care must be taken of the input matrix. There is a subtle issue on soundness which we address in Section 7.1.3. Finally, we combine everything in Section 7.1.4.

The query complexity of the PCPP in Section 7.1 is $T(n)^{\Omega(1)}$ (if we want the proof length to be at most $T(n) \cdot \mathrm{polylog}(T(n))$), but ideally we want a PCPP with constant query complexity. Thus, in Section 7.2, we compose it with another PCPP in [Mie09] to obtain a rectangular PCPP with constant query complexity, while maintaining proof length $T(n) \cdot \mathrm{polylog}(T(n))$. (Note that it is *not* required that the PCPP in [Mie09] is also rectangular.) The main body of Section 7.2 proves a composition theorem for rectangular PCPPs; the final PCPP with constant query complexity is constructed in Section 7.2.1.

In this section, $\mathsf{NTIME}[T(n)]$ always refers to $\mathsf{NTIME}_{\mathrm{TM}}[T(n)]$, i.e., we only consider the Turing machine model.

7.1 The Rectangular PCPP in [BGH+05]

In this sub-section, we review the PCPP for $\mathsf{NTIME}[T(n)]$ constructed in [BGH+06, BGH+05], with an emphasis on its rectangularity. The properties of the PCPP in [BGH+05, BGH+06] (including rectangularity) are summarised in Theorem 7.1.

Recall that a function $f : \mathbb{N} \to \mathbb{N}$ is *good* if given the input n in binary, we can compute $f(n)$ (also in binary) in time $\mathrm{poly}(\log n, \log f(n))$.

Theorem 7.1. *The following holds for all constants $m \geq 1$ and $s, \delta > 0$. Let $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $n \leq T(n) \leq 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \leq \log T(n)$, and $w_{\mathsf{input}}(n) \leq \log n$. Let*

$$h_{\mathsf{proof}}(n) := \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}(n), \qquad \text{and}$$
$$h_{\mathsf{input}}(n) := \lceil \log n \rceil - w_{\mathsf{input}}(n).$$

Moreover, suppose that for some large enough constant $C \geq 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \leq 1 - \frac{C \log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$, $H_{\mathsf{proof}}(n) := 2^{h_{\mathsf{proof}}(n)}$, $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$, and $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$. Then $\mathsf{NTIME}[T(n)]$ has a robust and rectangular PCP of proximity with an $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix and an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix, whose other parameters are specified in Table 2.

Moreover, V_{row} and V_{col} are projections over seed.row *and* seed.col *respectively, computable in polynomial time given* seed.shared.

Soundness error	s
Proximity parameter	δ
Robustness parameter	$\Omega(\delta)$
Row randomness	$h_{\mathsf{proof}} - (2/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (4/m)\log T(n)$
Shared randomness	$(6/m)\log T(n) + O(\log\log T(n))$
Query complexity	$T(n)^{1/m} \cdot \mathrm{polylog}(T(n))$
Decision complexity	

Table 2: Parameters of the PCPP constructed in Theorem 7.1.

We need an efficient construction of small-biased sets. In particular, let $\lambda > 0$, $m \in \mathbb{N}$, and $\mathbb{F}$ be a finite field of characteristic 2, we need a λ-biased set $S_\lambda \subseteq \mathbb{F}^m$. Besides being λ-biased, S_λ should possess an additional property: for every element $\vec{y} \in S_\lambda$, the first coordinate of $\vec{y}$ is non-zero. It is claimed in [BHPT20] that (under suitable conditions) for any $(\lambda/4)$-biased set $S_{\lambda/4}$, if we remove every element $\vec{y} \in S_{\lambda/4}$ with $y_1 = 0$, then the remaining set is still λ-biased. For completeness we provide a proof for the above claim in Appendix A.6.

Lemma 7.2. *Let $\lambda < 0.1$, q, m be integers such that $q \geq \log \frac{4}{\lambda}$, and let $\mathbb{F} = \mathrm{GF}(2^q)$. There is a deterministic polynomial-time algorithm that on input $(1^m, 1^q, 1^{\lceil 1/\lambda \rceil})$, outputs a λ-biased set $S_\lambda \subseteq (\mathbb{F} \setminus \{0\}) \times \mathbb{F}^{m-1}$ of size $O((qm/\lambda)^2)$.*

7.1.1 The PCPP Verifier

Let $L \in \mathsf{NTIME}[T(n)]$, we describe the PCPP verifier for L. The PCPP verifier receives two oracles: the input oracle Π_{input} (consisting of the input in verbatim) and the proof oracle Π_{proof}.

Set up. Let α be a universal constant as defined in the proof of [BGH+05, Theorem 6.4]. We set the following parameters:

$$\begin{aligned} t &:= \log T(n), \\ h &:= \lceil (t+3)/m \rceil, \\ f &:= h + \alpha \log_2 t, \\ \lambda &:= 1/ct, && \text{for some universal constant } c. \end{aligned}$$

We work with the field $\mathbb{F} := \mathrm{GF}(2^f)$. We treat $\mathbb{F}$ as a vector space of dimension f over $\mathrm{GF}(2)$, and let $e_1, \ldots, e_f$ be its basis. Each element $v \in \mathbb{F}$ can then be written as $v = \sum_{i=1}^{f} e_i b_i$ for $b_i \in \mathrm{GF}(2)$, and we denote the binary representation of v as $\mathsf{bin}(v) = (b_1 b_2 \ldots b_f)$. Let H be the vector space spanned by $e_1, e_2, \ldots, e_h$. We also define two bijections

$$\mathsf{bin}_{H^m} : H^m \to \{0,1\}^{hm} \quad \text{and} \quad \mathsf{bin}_{\mathbb{F}^m} : \mathbb{F}^m \to \{0,1\}^{fm}.$$

The bijection $\mathsf{bin}_{\mathbb{F}^m}$ is the usual one: it maps $(b_1 e_1 + \cdots + b_f e_f, b_{f+1} e_1 + \cdots + b_{2f} e_f, \ldots, b_{(m-1)f+1} e_1 + \cdots + b_{mf} e_f)$ to $(b_1 b_2 \ldots b_{mf})$. We also treat binary strings as numbers where the leftmost bit is the least significant one and the rightmost bit is the most significant one. For example, the string $(b_1 b_2 \ldots b_{mf})$ is treated as $\sum_{i=1}^{mf} b_i 2^{i-1}$.

We use the following injection $I_t : [n] \to H^m$ to project the input to H^m:

$$I_t(i) = \mathsf{bin}_{H^m}^{-1}(2^{t+1} + i). \tag{4}$$

That is, the i-th input bit will be embedded into the position $I_t(i) \in H^m$. Define the set $I := \{I_t(k) : k \le |\Pi_{\mathsf{input}}|\}$, then the input will be stored on the index set I.

Note that we have not specified bin_{H^m} yet; the definition of bin_{H^m} is a bit technical and we will specify it later. The reason is that we want to map the input to a matrix of dimension $H_{\mathsf{input}} \times W_{\mathsf{input}}$. More specifically, the input string occupies positions $[2^{t+1}+1, 2^{t+1}+n]$, and we want to map this portion into a subset of H^m which corresponds to a rectangle of dimensions $H_{\mathsf{input}} \times W_{\mathsf{input}}$.

Remark 7.3. The definition of I_t (i.e., Eq. (4)) is derived from [BGH+05] as follows.

1. First, we reduce L to the Generalized deBruijn Graph Coloring problem ([BGH+05, Definition 4.3]). The i-th bit of Π_{input} is mapped to the $(2^{t+1}+i)$-th node of the first layer.
2. Then, we reduce the above coloring problem to the Multivariate Algebraic Constraint Satisfaction Problem ([BGH+05, Definition 6.4]). In this step, for every i, the i-th node in (the first layer of) the deBruijn graph is mapped to the vector $\mathsf{bin}_{H^m}^{-1}(i) \in H^m$.

Combining the above two steps, it follows that the i-th bit of Π_{input} is mapped to $I_t(i)$.

The PCPP proof will have length $|\mathbb{F}|^m \cdot \ell$ for some $\ell = \mathrm{polylog}(T)$; we treat it as an oracle $\Pi_{\mathsf{proof}} : \mathbb{F}^m \to \{0,1\}^\ell$.[27] Without loss of generality we may assume ℓ is a power of 2. The i-th bit of the proof (viewed as a string of length $|\mathbb{F}|^m \cdot \ell$) is equal to the k-th bit of $\Pi_{\mathsf{proof}}[\mathsf{bin}_{\mathbb{F}^m}^{-1}(j)]$, where $j := \lfloor i/\ell \rfloor$ and $k := i \bmod \ell$.

Lines. A *line* $\mathcal{L}$ over $\mathbb{F}^m$ is a set of the form $\{\vec{x} + t\vec{y} : t \in \mathbb{F}\}$. Here $\vec{x} \in \mathbb{F}^m$ is called the *intercept* of $\mathcal{L}$ and $\vec{y} \in \mathbb{F}^m$ is called the *direction* of $\mathcal{L}$. The PCPP verifier will make queries along the following two types of lines over $\mathbb{F}^m$:

[27] Actually, in [BGH+05], each entry $\Pi_{\mathsf{proof}}(\vec{x})$ is an error-corrected version of a vector in $\mathbb{F}^{\mathrm{polylog}(T)}$. The use of error correcting codes ensures that the PCPP verifier is robust.

- A *first-axis parallel line* is a line where $\vec{y} = (1, 0, 0, \ldots, 0)$ and $\vec{x} \in \mathbb{F}^m$. To sample a uniform first-axis parallel line, it suffices to choose $\vec{x}$ from $\{0\} \times \mathbb{F}^{m-1}$ uniformly at random using $(m-1)\log(|\mathbb{F}|)$ bits of randomness.
- Fix a λ-biased set $S_\lambda \subseteq \mathbb{F}^m$ constructed in Lemma 7.2. A *pseudorandom line* is a line where $\vec{x} \in \mathbb{F}^m$ and $\vec{y} \in S_\lambda$. Each line has $|\mathbb{F}|$ different representations (since the intercepts $\vec{x} + t\vec{y}$ represent the same line for all $t \in \mathbb{F}$). Therefore, we specify a *canonical* representation for each line.

  To sample a pseudorandom line in the canonical way, we first choose $\vec{y}$ from S_λ uniformly at random, and then sample $\vec{x}$ from $\{0\} \times \mathbb{F}^{m-1}$ uniformly at random. (Note that the first coordinate of $\vec{y}$ is always non-zero, therefore every pseudorandom line intersects $\{0\} \times \mathbb{F}^{m-1}$.) This uses $\log(|S_\lambda|) + (m-1)\log(|\mathbb{F}|)$ bits of randomness.

Query pattern. To verify rectangularity, it suffices to describe the query pattern of the verifier, i.e., the entries of Π_{input} and Π_{proof} that are queried for a given randomness.

Let seed be the randomness of the verifier, which has length $\log\big(|S_\lambda| \cdot |\mathbb{F}|^{m-1}\big)$. We partition seed into

$$\mathsf{seed} := (R_2, R_3, \ldots, R_m, R_y),$$

where each R_i $(2 \le i \le m)$ has length $\log|\mathbb{F}|$ and corresponds to an element in $\mathbb{F}$, and R_y has length $\log|S_\lambda|$ and corresponds to an element in S_λ. Then seed determines a first-axis parallel line $\mathcal{L}_0$ and a canonical pseudorandom line $\mathcal{L}_1$ as follows. The intercepts of both $\mathcal{L}_0$ and $\mathcal{L}_1$ are $\vec{x} = (0, R_2, R_3, \ldots, R_m)$; the direction of $\mathcal{L}_0$ is $(1, 0, 0, \ldots, 0)$, and the direction of $\mathcal{L}_1$ is the R_y-th element of S_λ.

Let $\mathsf{shift} : \mathbb{F}^m \to \mathbb{F}^m$ denote the cyclic shift one step to the left; i.e.,

$$\mathsf{shift}(a_1, a_2, \ldots, a_m) = (a_2, a_3, \ldots, a_m, a_1)$$

for a line $\mathcal{L}$, $\mathsf{shift}(\mathcal{L})$ denotes the set $\{\mathsf{shift}(x) : x \in \mathcal{L}\}$.

Now we are ready to describe the query pattern of the PCPP verifier:

- For every point $\vec{x}$ on $\mathcal{L}_0$, $\mathsf{shift}(\mathcal{L}_0)$, and $\mathcal{L}_1$, it makes a query to $\Pi_{\mathsf{proof}}[\vec{x}]$.
- For every $\vec{x} \in \mathcal{L}_1 \cap I$, it also makes a query to $\Pi_{\mathsf{input}}[I_t^{-1}(\vec{x})]$.

Remark 7.4. This query pattern is derived from [BGH+06] (see also [BHPT20, Section C]). In particular:

- Robust Low-Degree Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_1]$;
- Robust Identity Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_0]$;
- Robust Edge-Consistency Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_0]$ and $\Pi_{\mathsf{proof}}[\mathsf{shift}(\mathcal{L}_0)]$;
- Robust Zero Propagation Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_0]$ and $\Pi_{\mathsf{proof}}[\mathsf{shift}(\mathcal{L}_0)]$;
- Robust Proximity Test makes queries to $\Pi_{\mathsf{proof}}[\mathcal{L}_1]$ and $\Pi_{\mathsf{input}}[I_t^{-1}(\mathcal{L}_1 \cap I)]$.

As Robust Proximity Test was not needed in, and not described by [BHPT20] (since they were constructing a PCP instead of a PCPP), we describe its details here. This test queries Π on every point in $\mathcal{L}_1$, and unbundles the answers to obtain the values of A_0 (a certain proof polynomial in [BGH+05, Definition 6.3]) on $\mathcal{L}_1$. Then it queries Π_{input} on every point $k \in \mathcal{L}_1 \cap I$, and checks whether $\Pi_{\mathsf{input}}[k] = f^t_{\mathsf{extract}}(A_0[I_t(k)])$ holds, where f^t_{extract} is a certain function defined in [BGH+05, Definition 6.3].

7.1.2 Rectangularity of the PCPP Verifier

Given the query pattern of the verifier described above, we now show that the verifier is rectangular. Recall that the verifier makes $3|\mathbb{F}|$ queries to Π_{proof}; let us call them $(\vec{a}_1, \ldots, \vec{a}_{|\mathbb{F}|})$, $(\vec{a}_{|\mathbb{F}|+1}, \ldots, \vec{a}_{2|\mathbb{F}|})$, and $(\vec{a}_{2|\mathbb{F}|+1}, \ldots, \vec{a}_{3|\mathbb{F}|})$, which are on $\mathcal{L}_0$, $\mathsf{shift}(\mathcal{L}_0)$, and $\mathcal{L}_1$ respectively. The following lemma shows that for each $1 \le j \le m$, the j-th coordinate of each query only depends on R_j, R_{j+1}, and R_y.

Lemma 7.5. *Fix the random string* $\mathsf{seed} = (R_2, R_3, \ldots, R_m, R_y)$*, and for convenience define* $R_1 = R_{m+1} = 0^{\log|\mathbb{F}|}$*. Denote each* $\vec{a}_i = (a_{i,1}, \ldots, a_{i,m}) \in \mathbb{F}^m$*. Then for every* $j \in [m]$*,* $(a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j})$ *only depends on* R_j*,* R_{j+1}*, and* R_y*.*

Moreover, for a fixed R_y*,* $(a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j})$ *is a projection over* (R_j, R_{j+1})*.*

Proof. Let $h_1, h_2, \ldots, h_{|\mathbb{F}|}$ be an enumeration of all the elements in $\mathbb{F}$. Then

$$\begin{aligned}(a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j}) = (&R_j + h_1 y_{1,j}, R_j + h_2 y_{1,j}, \ldots, R_j + h_{|\mathbb{F}|} y_{1,j},\\ &R_{j+1} + h_1 y_{2,j}, R_{j+1} + h_2 y_{2,j}, \ldots, R_{j+1} + h_{|\mathbb{F}|} y_{2,j},\\ &R_j + h_1 y_{3,j}, R_j + h_2 y_{3,j}, \ldots, R_j + h_{|\mathbb{F}|} y_{3,j}) \end{aligned} \tag{5}$$

where $\vec{y}_1, \vec{y}_2, \vec{y}_3 \in \mathbb{F}^m$ is defined to be $\vec{y}_1 = (1, 0, 0, \ldots, 0)$, $\vec{y}_2 = (0, 0, \ldots, 0, 1)$ and $\vec{y}_3$ is R_y-th vector in S_λ.

It is easy to see that Eq. (5) only depends on R_j, R_{j+1}, and R_y. Moreover, each coordinate $a_{i,j}$ is the sum of R_j or R_{j+1} with an element of the form $h_k y_{l,j}$. Note that addition over $\mathbb{F} = \mathrm{GF}(2^f)$ is equivalent to bitwise XOR over $\{0,1\}^f$, it follows that Eq. (5) is indeed a projection over (R_j, R_{j+1}). □

Definition of bin_{H^m}. Now we define bin_{H^m}. Roughly speaking, the goal of this definition is to "shape" the input oracle as a rectangle of size $H_{\mathsf{input}} \times W_{\mathsf{input}}$.

We treat every element in H as a binary string of length h. For a vector $\vec{a} = (a_1, a_2, \ldots, a_m) \in H^m$, the natural encoding of $\vec{a}$ is the concatenation of $a_1, a_2, \ldots, a_m$ (from the lowest bits to the highest bits), where each a_i is treated as an element in $\{0,1\}^h$. We denote this encoding as $\mathsf{bin}^\circ \in \{0,1\}^{mh}$. Let $k := \lceil (w_{\mathsf{proof}} - \log \ell) \cdot (h/f) \rceil$, as we will show later, the lowest k bits of bin° are computed by V_{col} and the rest bits of bin° are computed by V_{row}. Then, we define $\mathsf{bin}_{H^m}(\vec{a})$ to be the concatenation of (from the lowest bits to the highest bits):

$$\mathsf{bin}^\circ[1, w_{\mathsf{input}}], \mathsf{bin}^\circ[k+1, k+h_{\mathsf{input}}], \mathsf{bin}^\circ[w_{\mathsf{input}}+1, k], \text{ and } \mathsf{bin}^\circ[k+h_{\mathsf{input}}+1, hm]. \tag{6}$$

Some intuitions behind the definition of bin_{H^m} is as follows. The lowest k bits of bin° are computed by V_{col} and the rest bits are computed by V_{row}. To make the input matrix size $H_{\mathsf{input}} \times W_{\mathsf{input}}$, among the lowest $\lceil \log n \rceil$ bits, there needs to be w_{input} bits computed by V_{col} and $\lceil \log n \rceil - w_{\mathsf{input}} = h_{\mathsf{input}}$ bits computed by V_{row}. In the definition, we simply put the lowest w_{input} bits computed by V_{col} and the lowest h_{input} bits computed by V_{row} as the lowest $\lceil \log n \rceil$ bits of $\mathsf{bin}_{H^m}(\vec{a})$, and put the rest bits as the highest bits of $\mathsf{bin}_{H^m}(\vec{a})$.

We define $c_1 := \lceil w_{\mathsf{input}}/h \rceil$, $c_2 := \lceil k/h \rceil$, $c_3 := \lceil (k + h_{\mathsf{input}})/h \rceil$. Then the w_{input}-th bit of bin° is in a_{c_1}, the k-th bit of bin° is in a_{c_2}, and the $(k + h_{\mathsf{input}})$-th bit of bin° is in a_{c_3}.

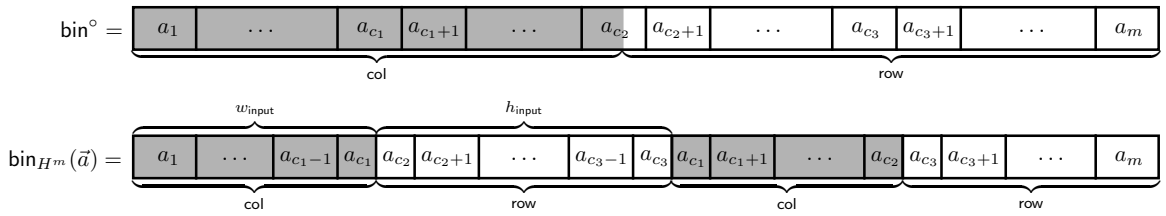

Figure 3: The bit-string $\mathsf{bin}_{H^m}(a_1, a_2, \ldots, a_m)$. In this figure, the leftmost bits are the *least* significant ones.

Note that

$$w_{\mathsf{input}} \le w_{\mathsf{proof}}(1 - \Theta(\log f)/f) \le k, \qquad \text{and}$$
$$k + h_{\mathsf{input}} \le (w_{\mathsf{proof}} - \log \ell) \cdot (h/f) + h_{\mathsf{proof}} \cdot (h/f) \le hm. \tag{7}$$

Here, Eq. (7) is because $w_{\mathsf{proof}} + h_{\mathsf{proof}} = \log |\Pi_{\mathsf{proof}}| = fm + \log \ell$. Since $w_{\mathsf{input}} \le k$ and $k + h_{\mathsf{input}} \le hm$, Eq. (6) is well-defined.

Partition of the random seed. We partition seed into:

$$\begin{aligned} \mathsf{seed.col} &= (R_1, R_2, \ldots, R_{c_2-1}), \\ \mathsf{seed.row} &= (R_{c_2+2}, R_{c_2+3}, \ldots, R_{m-1}), \text{ and} \\ \mathsf{seed.shared} &= (R_{c_2}, R_{c_2+1}, R_y). \end{aligned}$$

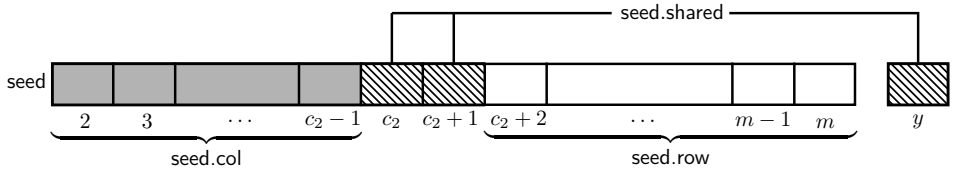

Figure 4: The partition of the random seed.

By Lemma 7.5, it suffices to know R_j, R_{j+1}, and R_y in order to calculate $(a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j})$. Therefore, V_{col} is able to calculate the $1, 2, \ldots, c_2$-th coordinates of each query; V_{row} is able to calculate the $c_2, c_2 + 1, \ldots, m$-th coordinates of each query.

In this partition, we have $|\mathsf{seed.col}| = (c_2 - 1)f \ge w_{\mathsf{proof}} - 2t/m$ and $|\mathsf{seed.row}| = (m - c_2 - 2)f \ge h_{\mathsf{proof}} - 4t/m$. For technical convenience, we will assume $|\mathsf{seed.col}| = w_{\mathsf{proof}} - 2t/m$ and $|\mathsf{seed.row}| = h_{\mathsf{proof}} - 4t/m$ from now on; the rest $6t/m + O(\log t) + \log |S_\lambda|$ random bits are in $\mathsf{seed.shared}$.

The predicates V_{type}, V_{row}, and V_{col}. Recall that we treat Π_{proof} as an oracle whose entries are length-ℓ strings, and we make $3|\mathbb{F}|$ queries to Π_{proof}. This means that we actually make $3|\mathbb{F}|\ell$ queries to the bit-string corresponding to Π_{proof}. We also make $|\mathbb{F}|$ queries to Π_{input}.

It is easy to describe V_{type}: the first $3|\mathbb{F}|\ell$ queries are to the proof oracle, and the last $|\mathbb{F}|$ queries are to the input oracle.

Now consider the i-th query where $1 \le i \le 3|\mathbb{F}|\ell$; these are queries made to Π_{proof}. Let $j := \lfloor (i-1)/\ell \rfloor$ and $k := (i-1) \bmod \ell$, then the i-th query probes the k-th bit of $\Pi_{\mathsf{proof}}[\vec{a}_j]$, where $\vec{a}_j \in \mathbb{F}^m$ is defined above. We want to specify V_{row} and V_{col} such that the index of the i-th query is

$$\mathsf{irow}[i] \cdot W_{\mathsf{proof}} + \mathsf{icol}[i] = \mathsf{bin}_{\mathbb{F}^m}(\vec{a}_j) \cdot \ell + k.$$

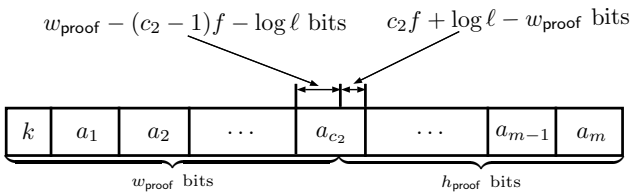

Figure 5: The binary representation of the address $\mathsf{bin}_{\mathbb{F}^m}(\vec{a}) \cdot \ell + k$. In this figure, the leftmost bits are the *least* significant ones. The lowest w_{proof} bits are outputted by V_{col}, while the rest bits are outputted by V_{row}.

Recall that V_{col} can compute $a_{j,1}, a_{j,2}, \ldots, a_{j,c_2}$ using Lemma 7.5. Then, it outputs $\mathsf{icol}[i]$ as the concatenation of k, $a_{j,1}, \ldots, a_{j,c_2-1}$ and the lowest $w_{\mathsf{proof}} - (c_2-1)f - \log \ell$ bits of a_{j,c_2}.[28] Similarly, V_{row} can compute $a_{j,c_2}, a_{j,c_2+1}, \ldots, a_{j,m}$. It outputs $\mathsf{irow}[i]$ as the concatenation of the highest $c_2 f + \log \ell - w_{\mathsf{proof}}$ bits of a_{j,c_2}, and a_{j,c_2+1}, a_{j,c_2+2}, $\ldots$, $a_{j,m}$. It follows from Lemma 7.5 that the first $3|\mathbb{F}|\ell$ entries of V_{row} and V_{col} are projections over $\mathsf{seed.row}$ and $\mathsf{seed.col}$, computable in polynomial time given $\mathsf{seed.shared}$.

Finally, we consider the $(i+3|\mathbb{F}|\ell)$-th query where $1 \le i \le |\mathbb{F}|$; these are queries made to Π_{input}. In particular, recall that the canonical pseudorandom line $\mathcal{L}_1$ consists of vectors $\vec{a}_{2|\mathbb{F}|+1}, \vec{a}_{2|\mathbb{F}|+2}, \ldots, \vec{a}_{3|\mathbb{F}|}$. If $\vec{a}_{2|\mathbb{F}|+i} \in I$ then we query the $I_t^{-1}(\vec{a}_{2|\mathbb{F}|+i})$-th bit of Π_{input}, otherwise we do nothing.

For notational convenience, we denote $\vec{a}^\star := \vec{a}_{2|\mathbb{F}|+i}$ and $\mathsf{bin} := \mathsf{bin}_{H^m}(\vec{a}^\star)$. Now our goal is to specify V_{row} and V_{col} such that the index of the $(i+3|\mathbb{F}|\ell)$-th query is

$$\mathsf{irow}\Big[i + 3|\mathbb{F}|\ell\Big] \cdot W_{\mathsf{input}} + \mathsf{icol}\Big[i + 3|\mathbb{F}|\ell\Big] = I_t^{-1}(\vec{a}^\star) = \mathsf{bin} - 2^{t+1}.$$

If either V_{row} or V_{col} outputs $\perp$, or $\mathsf{bin} - 2^{t+1} \notin [0, n)$, then we do not make this query.

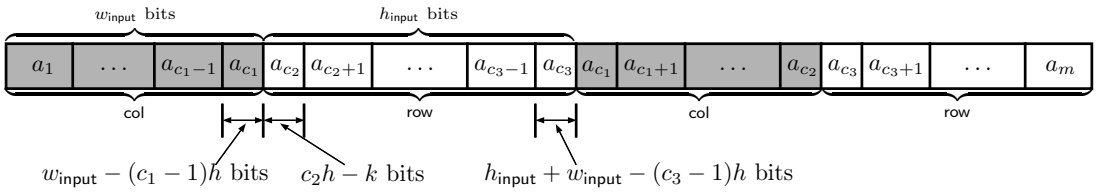

Figure 6: The bit-string $\mathsf{bin}_{H^m}(\vec{a})$. Again, the leftmost bits are the *least* significant ones.

Recall that V_{col} can compute $a_1^\star, a_2^\star, \ldots, a_{c_2}^\star$.

- If any of these elements are not in H, it outputs $\perp$.
- If any of the elements $a_{c_1+1}^\star, \ldots, a_{c_2}^\star$ is non-zero, then $\mathsf{bin} \bmod 2^t$ is not in the range $[0, n)$, and it outputs $\perp$.

[28] Note that $c_2 = \lceil (w_{\mathsf{proof}} - \log \ell)/f \rceil$, which means the "dividing point" between w_{proof} and h_{proof} is in a_{c_2}.

- Otherwise, the concatenation of $a^\star_1$, $a^\star_2$, …, $a^\star_{c_1-1}$, and the lowest $(w_{\mathsf{input}} - (c_1-1)h)$ bits of $a^\star_{c_1}$ is equal to $(\mathsf{bin} \bmod W_{\mathsf{input}})$. In this case it outputs

$$\mathsf{icol}\Big[i + 3|\mathbb{F}|\ell\Big] = \mathsf{bin} \bmod W_{\mathsf{input}}.$$

Also recall that V_{row} can compute $a^\star_{c_2}, a^\star_{c_2+1}, \ldots, a^\star_m$.

- If any of these elements are not in H, it outputs $\perp$.
- If any of the elements $a^\star_{c_3+1}$, $a^\star_{c_3+2}$, …, $a^\star_{m-1}$ is non-zero, or $a^\star_m \neq 2^{t+1-(m-1)h}$, then bin is not of the form $2^{t+1} + i$ $(0 \le i < n)$, and it outputs $\perp$.[29]
- Otherwise, the concatenation of the lowest $(c_2 h - k)$ bits of $a^\star_{c_2}$, and $a^\star_{c_2+1}$, $a^\star_{c_2+2}$, …, $a^\star_{c_3-1}$, and the lowest $(h_{\mathsf{input}} + w_{\mathsf{input}} - (c_3-1)h)$ bits of $a^\star_{c_3}$, is equal to $\lfloor(\mathsf{bin} - 2^{t+1})/W_{\mathsf{input}}\rfloor$. In this case it outputs

$$\mathsf{irow}\Big[i + 3|\mathbb{F}|\ell\Big] = \lfloor(\mathsf{bin} - 2^{t+1})/W_{\mathsf{input}}\rfloor.$$

7.1.3 Robust Soundness Amplification

The above PCPP only satisfies a weak version of *expected robust soundness*: the *expected* fraction of bits that we need to flip in order to make the verifier accept is at least ρ, where the expectation is over the choice of seed. Ideally, we want the following stronger version of robust soundness: with probability at most s (where s can be made arbitrarily small), it is possible to flip a ρ fraction of bits read by the verifier to make it accept.

We can convert expected robust soundness to standard robust soundness (with an arbitrarily small soundness parameter s) using expander walks. Take a constant-degree expander graph $G = (V, E)$ where $V = \{0,1\}^{r(n)}$. Use $r(n) + O(1)$ random bits to sample a random walk of length $O(1)$ over G. Then, for every vertex $v \in \{0,1\}^{r(n)}$ in the random walk, run the old PCPP verifier with v as randomness, reject if the old PCPP verifier rejects. If every invocation of the old PCPP verifier accepts, then our new PCPP verifier also accepts.

However, we need to be careful when implementing this approach: To preserve the rectangularity of the verifier, we take the *tensor product* of three expanders (one for row randomness, one for column randomness, and one for shared randomness). To ensure that V_{row} and V_{col} are (still) projections, we use the following family of 1-*local* expanders:

Lemma 7.6 ([VW18])**.** *For every $\lambda < 1$, there is some $d = \mathrm{poly}(1/\lambda)$ such that the following holds. For every integer n, there are d explicit* projections *(i.e., NC^0_1 circuits) $C_1, C_2, \ldots, C_d : \{0,1\}^n \to \{0,1\}^n$ such that the following graph G_n is an expander graph with second largest eigenvalue at most λ: The vertex set of G_n is $\{0,1\}^n$, and each vertex $x \in \{0,1\}^n$ has d neighbors $C_1(x), C_2(x), \ldots, C_d(x)$.*

Theorem 7.7. *Let L be a language. Suppose that L has a robust and rectangular PCP of proximity V^{old} with parameters specified in Table 3. Then for every parameter $s = s(n)$, L has a robust and rectangular PCP of proximity V^{new} with parameters specified in Table 3.*

Moreover, if $V^{\mathsf{old}}_{\mathsf{row}}$ and $V^{\mathsf{old}}_{\mathsf{col}}$ are projections computable in polynomial time given $\mathsf{seed}^{\mathsf{old}}.\mathsf{shared}$, then $V^{\mathsf{new}}_{\mathsf{row}}$ and $V^{\mathsf{new}}_{\mathsf{col}}$ are also projections computable in polynomial time given $\mathsf{seed}^{\mathsf{new}}.\mathsf{shared}$.

[29] Note that the $(t+1)$-st bit of $\mathsf{bin}_{H^m}(\vec{a}^\star)$ is indeed located in $a^\star_m$, since $hm-(t+1) \le (t+3)+m-(t+1) = m+2 < h$ when t is large enough. Another minor detail is that if $c_1 = m$, then the test that $a^\star_m = 2^{t+1-(m-1)h}$ should be performed by V_{col} instead of V_{row}.

Verifier	V^{old}	V^{new}
Proximity parameter	δ	δ
Soundness error	expected ρ	s
Robustness parameter		$\rho/3$
Row randomness	r_{row}	r_{row}
Column randomness	r_{col}	r_{col}
Shared randomness	r_{shared}	$r_{\mathsf{shared}} + O(\rho^{-2}\log(1/s)\log(1/\rho))$
Query complexity	q	$O(q \cdot \rho^{-2}\log(1/s))$
Decision complexity	d	$O(d \cdot \rho^{-2}\log(1/s) + \mathrm{poly}(r_{\mathsf{row}}, r_{\mathsf{col}}, r_{\mathsf{shared}}))$

Table 3: The parameters of the PCPPs in Theorem 7.7.

Proof. Let $\lambda := \rho/3$, G_{row} be the 1-local expander on vertex set $\{0,1\}^{r_{\mathsf{row}}(n)}$ whose second largest eigenvalue is at most λ, specified by Lemma 7.6; similarly let G_{col} be the 1-local expander on vertex set $\{0,1\}^{r_{\mathsf{col}}(n)}$, and G_{shared} be the 1-local expander on vertex set $\{0,1\}^{r_{\mathsf{shared}}(n)}$. Then G_{row}, G_{col}, and G_{shared} are d-regular graphs for $d := \mathrm{poly}(1/\delta)$. Let $G := G_{\mathsf{row}} \times G_{\mathsf{col}} \times G_{\mathsf{shared}}$ be the *tensor product* of the three expanders. That is, $G = (V, E)$ where

$$V := \{0,1\}^{r^{\mathsf{old}}(n)} = \{0,1\}^{r_{\mathsf{row}}(n)} \times \{0,1\}^{r_{\mathsf{col}}(n)} \times \{0,1\}^{r_{\mathsf{shared}}(n)}, \quad \text{and}$$
$$E := \{((u,v,w),(u',v',w')) : (u,u') \in E(G_{\mathsf{row}}), (v,v') \in E(G_{\mathsf{col}}), (w,w') \in E(G_{\mathsf{shared}})\}.$$

It follows from the properties of tensor product that the second largest eigenvalue of G is also at most λ.

Let $\ell := O(\lambda^{-2}\log(1/s))$. We take a length-$\ell$ random walk on G to generate ℓ random seeds $\mathsf{seed}_1, \ldots, \mathsf{seed}_\ell \in \{0,1\}^{r^{\mathsf{old}}(n)}$, and plug them into the old PCPP verifier one by one. An equivalent way to state this (from which the rectangularity is easier to see) is as follows. We add $3\lceil(\ell-1)\log d\rceil$ random bits into $\mathsf{seed.shared}$.

- The first $\lceil(\ell-1)\log d\rceil$ random bits correspond to $(\ell-1)$ numbers $\sigma_1, \sigma_2, \ldots, \sigma_{\ell-1} \in [d]$, which specify a random walk $(u_1, u_2, \ldots, u_\ell)$ in G_{row}: Namely, $u_1 = \mathsf{seed.row}$, and for every $2 \le i \le \ell$, $u_i = C_{\sigma_i}(u_{i-1})$. It is easy to see that each u_i is a projection over $\mathsf{seed.row}$, computable in polynomial time given $\mathsf{seed.shared}$.
- The middle $\lceil(\ell-1)\log d\rceil$ random bits specify a random walk $(v_1, v_2, \ldots, v_\ell)$ in G_{col}, starting from $\mathsf{seed.col}$. Each v_i is a projection over $\mathsf{seed.col}$, computable in polynomial time given $\mathsf{seed.shared}$.
- The last $\lceil(\ell-1)\log d\rceil$ random bits specify a random walk $(w_1, w_2, \ldots, w_\ell)$ in G_{shared}.

Then, our new PCPP verifier accepts seed if and only if for every $i \in [\ell]$, the old PCPP verifier accepts the following random seed:

$$(\mathsf{seed}^{\mathsf{old}}.\mathsf{row} = u_i, \mathsf{seed}^{\mathsf{old}}.\mathsf{col} = v_i, \mathsf{seed}^{\mathsf{old}}.\mathsf{shared} = w_i).$$

The query and decision complexity of our new PCPP verifier is easy to see. Fix $\mathsf{seed}^{\mathsf{new}}.\mathsf{shared}$, the row indices of each query only depends on $(u_1, u_2, \ldots, u_\ell)$, which only depends on $\mathsf{seed.row}$. Similarly, the column indices of each query only depends on $(v_1, v_2, \ldots, v_\ell)$, which only depends on $\mathsf{seed.col}$. Therefore the new PCPP verifier is rectangular. Moreover, each u_i can be computed by a projection over $\mathsf{seed.row}$, and each v_i can be computed by a projection over $\mathsf{seed.col}$.

Finally, we examine the robust soundness of our new PCPP. Fix an input oracle Π_{input} which is δ-far from L. Let $\mathsf{seed} \in V(G) = \{0,1\}^{r^{\mathsf{old}}(n)}$ be some randomness for the old PCPP verifier, we define $\delta_{\mathsf{change}}(\mathsf{seed})$ to be the fraction of bits read by the old PCPP verifier that we need to change in order to make the old PCPP verifier accept. Let $\mu := \mathbb{E}_{\mathsf{seed}}[\delta_{\mathsf{change}}(\mathsf{seed})]$, then $\mu \geq \rho$. By expander Chernoff bound (see, e.g., [Vad12, Theorem 4.22]), we have

$$\Pr\left[\left|\frac{1}{\ell}\sum_{i=1}^{\ell}\delta_{\mathsf{change}}(u_i,v_i,w_i) - \mu\right| \geq 2\lambda\right] \leq 2\exp(-\Omega(\lambda^2\ell)) \leq s.$$

It follows that w.p. at least $1-s$, we need to change $\geq \mu - 2\lambda \geq \rho/3$ fraction of bits read by the new PCPP verifier in order to make it accept. That is, the new PCPP verifier has soundness parameter s and robustness parameter $\rho/3$. □

7.1.4 Proof of Theorem 7.1

Now we prove Theorem 7.1.

Theorem 7.1. *The following holds for all constants $m \geq 1$ and $s, \delta > 0$. Let $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $n \leq T(n) \leq 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \leq \log T(n)$, and $w_{\mathsf{input}}(n) \leq \log n$. Let*

$$h_{\mathsf{proof}}(n) := \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}(n), \qquad \text{and}$$
$$h_{\mathsf{input}}(n) := \lceil \log n \rceil - w_{\mathsf{input}}(n).$$

Moreover, suppose that for some large enough constant $C \geq 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \leq 1 - \frac{C\log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$, $H_{\mathsf{proof}}(n) := 2^{h_{\mathsf{proof}}(n)}$, $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$, and $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$. Then $\mathsf{NTIME}[T(n)]$ has a robust and rectangular PCP of proximity with an $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix and an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix, whose other parameters are specified in Table 2.

Moreover, V_{row} and V_{col} are projections over $\mathsf{seed.row}$ and $\mathsf{seed.col}$ respectively, computable in polynomial time given $\mathsf{seed.shared}$.

Proof Sketch. Consider the PCPP verifier introduced in Sections 7.1.1 and 7.1.2. Its decision complexity, query complexity, and perfect completeness are shown in [BGH+06, BGH+05]. The rectangularity of the PCPP verifier is shown in Section 7.1.2. Note that $|S_\lambda| = O((fm/\lambda)^2)$, thus $\log|S_\lambda| = O(\log\log T(n))$. Recall that

$$\begin{aligned} r_{\mathsf{row}}(n) &= w_{\mathsf{proof}} - 2t/m, \\ r_{\mathsf{col}}(n) &= h_{\mathsf{proof}} - 4t/m, \\ r_{\mathsf{shared}}(n) &= 6t/m + O(\log t). \end{aligned}$$

The above PCPP verifier only achieves the following version of robustness property (see [BGH+06, Lemma 8.11]). Let $\delta_{\mathsf{proof}}(\mathsf{seed})$ denote the fraction of bits of Π_{proof} read by the verifier that we need to flip, if we want the verifier to accept given seed as randomness; $\delta_{\mathsf{input}}(\mathsf{seed})$ is defined analogously for Π_{input}. There is a constant $\rho > 0$ such that for every constant $\delta > 0$, if Π_{input} is δ-far from L, then for any proof oracle Π_{proof}, either $\mathbb{E}_{\mathsf{seed}}[\delta_{\mathsf{proof}}(\mathsf{seed})] \geq \rho$ or $\mathbb{E}_{\mathsf{seed}}[\delta_{\mathsf{input}}(\mathsf{seed})] \geq \delta/3$.

Recall that we make $n_{\mathsf{input}} := |\mathbb{F}|$ queries to Π_{input} and $n_{\mathsf{proof}} := 3|\mathbb{F}|\ell$ queries to Π_{proof}. We repeat each input query $n' := 9(\rho/\delta)\ell$ times. Now, if Π_{input} is δ-far from L, then the expected fraction of bits read by the verifier needs to be flipped is at least

$$\frac{\min\{\rho \cdot n_{\mathsf{proof}}, (\delta/3) \cdot n' \cdot n_{\mathsf{input}}\}}{n' \cdot n_{\mathsf{input}} + n_{\mathsf{proof}}} = \frac{\rho}{1 + 3(\rho/\delta)} = \Omega(\delta),$$

in order to make the verifier accept. Now, we have a PCPP verifier that achieves expected robustness $\Omega(\delta)$: the expected fraction of bits read by the verifier that needs to be flipped, in order to make the verifier accept, is at least $\Omega(\delta)$, where the expectation is over the random seed of the verifier. The decision and query complexities remain asymptotically the same.

Finally, we convert expected robustness to standard robustness via Theorem 7.7. For any parameter $s > 0$, we can reduce the soundness error to be at most s while preserving robustness error $\Omega(\delta)$, at the cost of adding $O(\log(1/s))$ random bits in $\mathsf{seed.shared}$ and paying an $O(\log(1/s))$ factor in the decision and query complexity. □

7.2 Composition and Final Construction

The PCPP verifier in Section 7.1 requires $T(n)^{\Omega(1)}$ query complexity. In this section, we compose it with an efficient PCPP to bring its query complexity down to $O(1)$. We assume basic familiarity with the composition theorem of PCPs; a good reference is [BGH+06, Section 2.4].

Let Circuit-Eval$^{\perp}$ denote the circuit value problem where the circuit has alphabet $\{0, 1, \perp\}$. Note that the input alphabet of the decision circuit of the PCPP constructed in Section 7.1 is $\{0, 1, \perp\}$ instead of $\{0, 1\}$. Therefore we should use a PCPP for Circuit-Eval$^{\perp}$ instead of Circuit-Eval for the inner PCPP.

We prove the following composition theorem.

Theorem 7.8. *Let $n \le T(n) \le 2^{\mathrm{poly}(n)}$. Suppose that $\mathsf{NTIME}[T(n)]$ has a robust and rectangular PCPP verifier V^{out} and Circuit-Eval$^{\perp}$ has a robust (but not necessarily rectangular) PCPP verifier V^{in} with parameters specified in Table 4. Moreover, assume that $\rho^{\mathsf{out}} \ge \delta^{\mathsf{in}}$, $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$,[30] $W_{\mathsf{proof}} \ge \ell^{\mathsf{in}}$, W_{proof} is a power of 2, and $r^{\mathsf{out}}_{\mathsf{col}} \le \log W_{\mathsf{proof}} \le r^{\mathsf{out}}_{\mathsf{col}} + r^{\mathsf{out}}_{\mathsf{shared}}$.*

Then $\mathsf{NTIME}[T(n)]$ has a robust and rectangular PCPP verifier V^{comp} with parameters specified in Table 4.

Proof. The composed PCPP is described in [BGH+06, Theorem 2.7], with one crucial difference in ROPs. In particular, the outer PCPP in [BGH+06] computes its decision predicate from its randomness, but the decision predicate of our outer PCPP takes (the encoded version of) its randomness as inputs.[31] For completeness, we present the composed PCPP in Algorithm 2.

The **for**-loop in Algorithm 2 computes the query indices of V^{comp} in detail. This **for**-loop will be very important as the rectangularity of V^{comp} relies on it. In the **for**-loop, we use the following notation:

[30] This is without loss of generality, because $\ell^{\mathsf{in}} \le 2^{r^{\mathsf{in}}} \cdot q^{\mathsf{in}}$, and in our case q^{in} will be a constant. We could always add $O(\log q^{\mathsf{in}}) = O(1)$ dummy bits to the inner verifier's randomness and pad the inner verifier's proof oracle to length $2^{r^{\mathsf{in}}}$.

[31] The reason to apply an error-correcting code on the randomness is that we want $\mathsf{Dec}^{\mathsf{out}}$ to have *robust* soundness. Let $(\Pi^{\mathsf{in}}, \mathsf{Enc}(\mathsf{seed}))$ be the input of $\mathsf{Dec}^{\mathsf{out}}$, if given seed, Π^{in} is far from being accepted by $\mathsf{Dec}^{\mathsf{out}}$, then $(\Pi^{\mathsf{in}}, \mathsf{Enc}(\mathsf{seed}))$ is also far from being accepted by $\mathsf{Dec}^{\mathsf{out}}$. This is not true if we do not encode seed.

Verifier	V^{out}	V^{in}	V^{comp}
Soundness error	s^{out}	s^{in}	$s^{\mathsf{out}} + s^{\mathsf{in}}$
Proximity parameter	δ^{out}	δ^{in}	δ^{out}
Robustness parameter	ρ^{out}	ρ^{in}	ρ^{in}
Row randomness	$r^{\mathsf{out}}_{\mathsf{row}}$	-	$r^{\mathsf{out}}_{\mathsf{row}}$
Column randomness	$r^{\mathsf{out}}_{\mathsf{col}}$	-	$r^{\mathsf{out}}_{\mathsf{col}} - r^{\mathsf{in}}$
Shared randomness	$r^{\mathsf{out}}_{\mathsf{shared}}$	r^{in}	$r^{\mathsf{out}}_{\mathsf{shared}} + 2 \cdot r^{\mathsf{in}}$
Proof matrix height	H_{proof}	-	$H_{\mathsf{proof}} + 2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W_{\mathsf{proof}}$
Proof matrix width	W_{proof}	-	W_{proof}
Query complexity	q^{out}	q^{in}	q^{in}
Parity check complexity	-	-	q^{in}
Decision complexity	d^{out}	d^{in}	d^{in}
Proof length	ℓ^{out}	ℓ^{in}	$\ell^{\mathsf{out}} + 2^{r^{\mathsf{out}}} \cdot \ell^{\mathsf{in}}$

Table 4: The parameters of the PCPPs in the composition theorem. Note that the input length of the inner PCPP is $d^{\mathsf{out}} = d^{\mathsf{out}}(n)$, e.g., r^{in} in the table actually refers to $r^{\mathsf{in}}(d^{\mathsf{out}}(n))$.

$$\begin{cases} i^{\mathsf{comp}}[k] \leftarrow i & \triangleright \text{ Regular Case} \\ \hline \mathsf{irow}^{\mathsf{comp}}[k] \leftarrow irow & \\ \mathsf{icol}^{\mathsf{comp}}[k] \leftarrow icol & \triangleright \text{ Rectangular Case} \end{cases}$$

This denotes that the k-th query index of V^{comp} is $(type, i)$ if there is no rectangularity constraint on V^{out}, and is $(type, irow, icol)$ if we require V^{out} and V^{comp} to be rectangular. It will always be the case that $irow = \lfloor i/W_{type} \rfloor$ and $icol = i \bmod W_{type}$ (for $type \in \{\mathsf{input}, \mathsf{proof}\}$), but it is helpful to explicitly spell out $irow$ and $icol$.

Now we specify the proof oracle of V^{comp}. Fix a choice of $\mathsf{seed}^{\mathsf{out}}$, let $\Pi^{\mathsf{in}}_{\mathsf{input}} := \Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}})$ be the input oracle of the inner PCPP verifier, specified in Eq. (8); let $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ be the PCPP proof for V^{in} of the statement that $\mathsf{Dec}^{\mathsf{out}}$ accepts $\Pi^{\mathsf{in}}_{\mathsf{input}}$. To construct the proof oracle of V^{comp}, we append the proofs $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ (for every possible choice of $\mathsf{seed}^{\mathsf{out}}$) at the end of $\Pi^{\mathsf{out}}_{\mathsf{proof}}$ (the proof oracle for the outer PCPP verifier). Let $N_{\mathsf{proof}} := W_{\mathsf{proof}}/\ell^{\mathsf{in}} = W_{\mathsf{proof}}/2^{r^{\mathsf{in}}}$, then every (appended) row contains N_{proof} proofs for the inner verifier. The final proof oracle will be an $H'_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix, where

$$H'_{\mathsf{proof}} := H_{\mathsf{proof}} + 2^{r^{\mathsf{out}}}/N_{\mathsf{proof}} = H_{\mathsf{proof}} + 2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W_{\mathsf{proof}}.$$

Partition of the random seed. The random seed is simply the concatenation of $\mathsf{seed}^{\mathsf{out}}$ and $\mathsf{seed}^{\mathsf{in}}$. The shared part of the random seed consists of three parts:

- $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$, the shared part of $\mathsf{seed}^{\mathsf{out}}$;
- $\mathsf{seed}^{\mathsf{in}}$, the randomness of the inner verifier;
- some arbitrary r^{in} bits of $\mathsf{seed}^{\mathsf{out}}.\mathsf{col}$ (the reason for adding these bits into $\mathsf{seed}.\mathsf{shared}$ will be clear when we discuss Line 22 of Algorithm 2).

Algorithm 2 The composed PCPP

Obtain the decision circuit $\mathsf{Dec}^{\mathsf{out}}$ of the outer verifier
Sample $\mathsf{seed}^{\mathsf{out}} \leftarrow \{0,1\}^{r^{\mathsf{out}}}$
Sample the queries $I^{\mathsf{out}} \leftarrow V^{\mathsf{out}}(\mathsf{seed}^{\mathsf{out}})$ of the outer verifier
Sample $\mathsf{seed}^{\mathsf{in}} \leftarrow \{0,1\}^{r^{\mathsf{in}}}$
Sample the queries $I^{\mathsf{in}} \leftarrow V^{\mathsf{in}}(\mathsf{seed}^{\mathsf{in}})$ of the inner verifier
Let $\mathsf{Dec}^{\mathsf{comp}} \leftarrow \mathsf{Dec}^{\mathsf{in}}$ be the decision circuit of the composed verifier
(Implicitly) define the input oracle of the inner verifier to be

$$\Pi^{\mathsf{in}}_{\mathsf{input}} := (\Pi_{\mathsf{input}}, \Pi_{\mathsf{proof}})|_{I^{\mathsf{out}}} \circ \mathsf{Enc}(\mathsf{seed}^{\mathsf{out}}) \tag{8}$$

for $k \leftarrow 1$ to q^{in} **do** ▷ Determine the position of the k-th query
  if $\mathsf{itype}^{\mathsf{in}}[k] = \mathsf{input}$ **then** ▷ This query is on $\Pi^{\mathsf{in}}_{\mathsf{input}}$
    if $i^{\mathsf{in}}[k] \le q^{\mathsf{out}}$ **then** ▷ This query is on $(\Pi_{\mathsf{input}}, \Pi_{\mathsf{proof}})|_{I^{\mathsf{out}}}$
      if $\mathsf{itype}^{\mathsf{out}}[i^{\mathsf{in}}[k]] = \mathsf{input}$ **then**
        $\mathsf{itype}^{\mathsf{comp}}[k] \leftarrow \mathsf{input}$
        $\begin{cases} i^{\mathsf{comp}}[k] \leftarrow i^{\mathsf{out}}[i^{\mathsf{in}}[k]] & \triangleright \text{ Regular Case} \\ \hline \mathsf{irow}^{\mathsf{comp}}[k] \leftarrow \mathsf{irow}^{\mathsf{out}}[i^{\mathsf{in}}[k]] \\ \mathsf{icol}^{\mathsf{comp}}[k] \leftarrow \mathsf{icol}^{\mathsf{out}}[i^{\mathsf{in}}[k]] & \triangleright \text{ Rectangular Case} \end{cases}$
      else
        $\mathsf{itype}^{\mathsf{comp}}[k] \leftarrow \mathsf{proof}$
        $\begin{cases} i^{\mathsf{comp}}[k] \leftarrow i^{\mathsf{out}}[i^{\mathsf{in}}[k]] & \triangleright \text{ Regular Case} \\ \hline \mathsf{irow}^{\mathsf{comp}}[k] \leftarrow \mathsf{irow}^{\mathsf{out}}[i^{\mathsf{in}}[k]] \\ \mathsf{icol}^{\mathsf{comp}}[k] \leftarrow \mathsf{icol}^{\mathsf{out}}[i^{\mathsf{in}}[k]] & \triangleright \text{ Rectangular Case} \end{cases}$
    else ▷ This query is on $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})$
      Fix the k-th input of $\mathsf{Dec}^{\mathsf{comp}}$ to be $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})[i^{\mathsf{in}}[k] - q^{\mathsf{out}}]$
  else ▷ This query is on the $\mathsf{seed}^{\mathsf{out}}$-th inner proof
    Let $N_{\mathsf{proof}} \leftarrow W_{\mathsf{proof}}/\ell^{\mathsf{in}}$ be the number of proof oracles for V^{in} in one row
    $\mathsf{itype}^{\mathsf{comp}}[k] \leftarrow \mathsf{proof}$
    $\begin{cases} i^{\mathsf{comp}}[k] \leftarrow \ell^{\mathsf{out}} + \mathsf{seed}^{\mathsf{out}} \cdot \ell^{\mathsf{in}} + i^{\mathsf{in}}[k] & \triangleright \text{ Regular Case} \\ \hline \mathsf{irow}^{\mathsf{comp}}[k] \leftarrow H_{\mathsf{proof}} + \lfloor \mathsf{seed}^{\mathsf{out}}/N_{\mathsf{proof}} \rfloor \\ \mathsf{icol}^{\mathsf{comp}}[k] \leftarrow \ell^{\mathsf{in}} \cdot (\mathsf{seed}^{\mathsf{out}} \bmod N_{\mathsf{proof}}) + i^{\mathsf{in}}[k] & \triangleright \text{ Rectangular Case} \end{cases}$

Formally, we split $\mathsf{seed}^{\mathsf{out}}.\mathsf{col}$ into $(\mathsf{seed}^{\mathsf{out}}.\mathsf{col}_1, \mathsf{seed}^{\mathsf{out}}.\mathsf{col}_2)$ where $|\mathsf{seed}^{\mathsf{out}}.\mathsf{col}_2| = r^{\mathsf{in}}$. We have

$$\begin{aligned}\mathsf{seed}.\mathsf{row} &= \mathsf{seed}^{\mathsf{out}}.\mathsf{row},\\ \mathsf{seed}.\mathsf{col} &= \mathsf{seed}^{\mathsf{out}}.\mathsf{col}_1,\\ \mathsf{seed}.\mathsf{shared} &= (\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}, \mathsf{seed}^{\mathsf{in}}, \mathsf{seed}^{\mathsf{out}}.\mathsf{col}_2).\end{aligned}$$

The predicates V_{type}, V_{row}, and V_{col}. These predicates can be inferred from the **for**-loop of Algorithm 2. In particular:

- Since $\mathsf{itype}^{\mathsf{out}}$ only depends on $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$, it is easy to see that $\mathsf{itype}^{\mathsf{comp}}$ only depends on $\mathsf{seed}.\mathsf{shared}$.
- It is easy to see that in Lines 13 and 16, $\mathsf{irow}^{\mathsf{comp}}$ only depends on $\mathsf{seed}.\mathsf{shared}$ and $\mathsf{seed}.\mathsf{row}$, and $\mathsf{icol}^{\mathsf{comp}}$ only depends on $\mathsf{seed}.\mathsf{shared}$ and $\mathsf{seed}.\mathsf{col}$.
- In Line 22, $\mathsf{icol}^{\mathsf{comp}}$ only depends on $\mathsf{seed}.\mathsf{shared}$ and the lowest $\log N_{\mathsf{proof}}$ bits of $\mathsf{seed}^{\mathsf{out}}$; $\mathsf{irow}^{\mathsf{comp}}$ only depends on $\mathsf{seed}.\mathsf{shared}$ and the highest $r^{\mathsf{out}} - \log N_{\mathsf{proof}}$ bits of $\mathsf{seed}^{\mathsf{out}}$. Note that $|\mathsf{seed}^{\mathsf{out}}.\mathsf{col}| \le \log W_{\mathsf{proof}} \le |\mathsf{seed}^{\mathsf{out}}.\mathsf{col}| + |\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}|$. Therefore

  $$\log N_{\mathsf{proof}} \le |\mathsf{seed}^{\mathsf{out}}.\mathsf{col}| + |\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}|, \qquad \text{and}$$
  $$r^{\mathsf{out}} - \log N_{\mathsf{proof}} \le r^{\mathsf{in}} + |\mathsf{seed}^{\mathsf{out}}.\mathsf{row}| + |\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}|.$$

  We can rearrange the bits of $\mathsf{seed}^{\mathsf{out}}$ such that the lowest $\log N_{\mathsf{proof}}$ bits belong to either $\mathsf{seed}^{\mathsf{out}}.\mathsf{col}_1$ or $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$ (thus either $\mathsf{seed}.\mathsf{col}$ or $\mathsf{seed}.\mathsf{shared}$), and the highest $r^{\mathsf{out}} - \log N_{\mathsf{proof}}$ bits belong to either $\mathsf{seed}^{\mathsf{out}}.\mathsf{row}$, $\mathsf{seed}^{\mathsf{out}}.\mathsf{col}_2$, or $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$ (thus either $\mathsf{seed}.\mathsf{row}$ or $\mathsf{seed}.\mathsf{shared}$). It follows that $\mathsf{irow}^{\mathsf{comp}}$ only depends on $\mathsf{seed}.\mathsf{row}$ and $\mathsf{seed}.\mathsf{shared}$, and $\mathsf{icol}^{\mathsf{comp}}$ only depends on $\mathsf{seed}.\mathsf{col}$ and $\mathsf{seed}.\mathsf{shared}$.

ROP with parity-check bits. How does $\mathsf{Dec}^{\mathsf{comp}}$ (the decision predicate of V^{comp}) depend on its randomness? Note that $\mathsf{Dec}^{\mathsf{comp}}$ is equal to $\mathsf{Dec}^{\mathsf{in}}$ except that in Line 18, we fix a certain input bit of $\mathsf{Dec}^{\mathsf{comp}}$ to be a certain bit in $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})$. Since Enc is a $\mathrm{GF}(2)$-*linear* error correcting code (Theorem 2.2), each bit of $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})$ is the XOR of a subset of indices in $\mathsf{seed}^{\mathsf{out}}$. (This is the reason that we need *parity-check* bits in ROP.) Also, $\mathsf{Dec}^{\mathsf{in}}$ only depends on $\mathsf{seed}.\mathsf{shared}$, therefore $\mathsf{Dec}^{\mathsf{comp}}$ only depends on $\mathsf{seed}.\mathsf{shared}$ and the q^{in} parity-check bits over $\mathsf{seed}.\mathsf{row}$ and $\mathsf{seed}.\mathsf{col}$. □

Corollary 7.9. *Suppose that in Theorem 7.8, for fixed* $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$, $V^{\mathsf{out}}_{\mathsf{row}}$ *and* $V^{\mathsf{out}}_{\mathsf{col}}$ *are projections over* $\mathsf{seed}^{\mathsf{out}}.\mathsf{row}$ *and* $\mathsf{seed}^{\mathsf{out}}.\mathsf{col}$ *(respectively), computable in polynomial time given* $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$. *Moreover, assume* H_{proof} *is a power of* 2.

Then for fixed $\mathsf{seed}^{\mathsf{comp}}.\mathsf{shared}$, $V^{\mathsf{comp}}_{\mathsf{row}}$ *and* $V^{\mathsf{comp}}_{\mathsf{col}}$ *are also projections over* $\mathsf{seed}^{\mathsf{comp}}.\mathsf{row}$ *and* $\mathsf{seed}^{\mathsf{comp}}.\mathsf{col}$, *computable in polynomial time given* $\mathsf{seed}^{\mathsf{comp}}.\mathsf{shared}$.

Proof. For each $k \le q^{\mathsf{in}}$, we execute the **for**-loop in Algorithm 2. If we execute Line 13 or Line 16, it is easy to see that $\mathsf{irow}^{\mathsf{comp}}[k]$ is a projection over $\mathsf{seed}.\mathsf{row}$ and $\mathsf{icol}^{\mathsf{comp}}[k]$ is a projection over $\mathsf{seed}.\mathsf{col}$. So it remains to examine Line 22:

- The lowest $r^{\mathsf{out}} - \log N_{\mathsf{proof}}$ bits of $\mathsf{irow}^{\mathsf{comp}}[k]$ are simply the highest $r^{\mathsf{out}} - \log N_{\mathsf{proof}}$ bits of $\mathsf{seed}^{\mathsf{out}}$; the higher bits of $\mathsf{irow}^{\mathsf{comp}}[k]$ are fixed to be H_{proof}.
- The lowest r^{in} bits of $\mathsf{icol}^{\mathsf{comp}}[k]$ are simply $i^{\mathsf{in}}[k]$; the higher bits of $\mathsf{icol}^{\mathsf{comp}}[k]$ are the lowest $\log N_{\mathsf{proof}}$ bits of $\mathsf{seed}^{\mathsf{out}}$. □

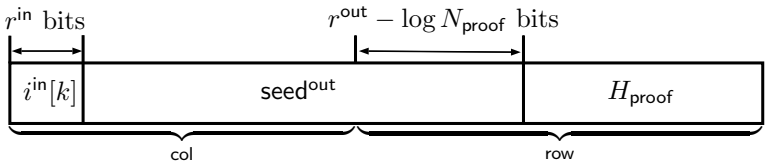

Figure 7: The bit-string $i^{\mathsf{comp}}[k]$ in Line 22. Again, the leftmost bits are the least significant ones.

7.2.1 The Final PCPP

We will use the constant query PCPP of [Mie09] for the inner verifier.

Theorem 7.10 ([Mie09])**.** *Let L be a pair language in $\mathsf{NTIME}[T(n)]$ for some non-decreasing function $T : \mathbb{Z}^+ \to \mathbb{Z}^+$. Then, for every constants $s, \delta > 0$, L has a PCPP verifier with parameters specified in Table 5.*

Proximity parameter	δ
Soundness error	s
Randomness complexity	$\log T(n) + O(\log \log T(n))$
Query complexity	$O(1)$
Decision complexity	$\mathrm{polylog}(T(n))$

Table 5: Parameters of the PCPP constructed in Theorem 7.10.

Now, we compose the outer PCPP (Theorem 7.1) and the inner PCPP (Theorem 7.10) to obtain the following:

Theorem 7.11. *The following holds for every constants $m \geq 1$ and $s, \delta > 0$. Let $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$, be good functions such that $n \leq T(n) \leq 2^{\mathrm{poly}(n)}$, $\frac{2}{m} \log T(n) \leq w_{\mathsf{proof}}(n) \leq \log T(n)$, and $w_{\mathsf{input}}(n) \leq \log n$. Let*

$$h_{\mathsf{proof}}(n) := \log T(n) + \Theta(m \log \log T(n)) - w_{\mathsf{proof}}(n), \qquad \text{and}$$
$$h_{\mathsf{input}}(n) := \lceil \log n \rceil - w_{\mathsf{input}}(n).$$

Moreover, suppose that for some large enough universal constant $C \geq 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \leq 1 - \frac{C \log \log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$, $H_{\mathsf{proof}}(n) := 2^{h_{\mathsf{proof}}(n)}$, $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$, and $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$. Then $\mathsf{NTIME}[T(n)]$ has a rectangular PCP of proximity with an $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix and an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix, whose other parameters are specified in Table 6.

Moreover, V_{row} and V_{col} are projections over $\mathsf{seed.row}$ and $\mathsf{seed.col}$ respectively, computable in polynomial time given $\mathsf{seed.shared}$.

Proximity parameter	δ
Soundness error	s
Total randomness	$\log T(n) + O(m \log\log T(n))$
Row randomness	$h_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (5/m)\log T(n)$
Shared randomness	$(10/m)\log T(n) + O(\log\log T(n))$
Query complexity	$O(1)$
Parity check complexity	$O(1)$
Decision complexity	$\mathrm{polylog}(T(n))$

Table 6: Parameters of the final PCPP.

Proof. Let V^{out} be the outer PCPP of Theorem 7.1 with soundness error $s^{\mathsf{out}} := s/2$, proximity parameter δ, and robustness parameter $\rho^{\mathsf{out}} := \Omega(\delta)$. The decision and query complexity of V^{out} are

$$d^{\mathsf{out}} = q^{\mathsf{out}} := T(n)^{1/m} \cdot \mathrm{polylog}(T(n)).$$

Let V^{in} be the inner PCPP of Theorem 7.10 with soundness parameter $s^{\mathsf{in}} := s/2$ and proximity parameter $\delta^{\mathsf{in}} := \rho^{\mathsf{out}}$. Plugging in $d^{\mathsf{out}} := T(n)^{1/m} \cdot \mathrm{polylog}(T(n))$, the randomness complexity of V^{in} becomes

$$r^{\mathsf{in}}(d^{\mathsf{out}}) = \frac{1}{m}\log T(n) + O(\log\log T(n)) + O(\log\log(1/s)).$$

It is easy to check that every technical condition in Theorem 7.8 hold. In particular

- $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}} \le T(n)^{1/m}\mathrm{polylog}(T(n)) < W_{\mathsf{proof}}$.
- $r^{\mathsf{out}}_{\mathsf{col}} + r^{\mathsf{out}}_{\mathsf{shared}} = (c_2 - 4)f + 4f + \log|S_\lambda| \ge c_2 f + 10 \ge w_{\mathsf{proof}}$; and
- $r^{\mathsf{out}}_{\mathsf{col}} = (c_2 - 4)f \le w_{\mathsf{proof}}$.

Now we invoke Theorem 7.8 to obtain the final PCPP with:

- Proximity parameter δ;
- soundness error $s^{\mathsf{out}} + s^{\mathsf{in}} = s$;
- proof matrix height $H_{\mathsf{proof}} + 2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W_{\mathsf{proof}} = \Theta(T(n)\log^{O(m)} T(n) s^{-1}/W_{\mathsf{proof}})$,
- query complexity and parity check complexity $O_s(1)$, and
- decision complexity $\mathrm{polylog}(T(n))$. □

Finally, we calculate the randomness complexity of the PCPP (note that $h^{\mathsf{comp}}_{\mathsf{proof}} \le h^{\mathsf{out}}_{\mathsf{proof}} + O(\log\log T(n))$):

- row randomness $r_{\mathsf{row}} = r^{\mathsf{out}}_{\mathsf{row}} = h^{\mathsf{out}}_{\mathsf{proof}} - (2/m)\log T(n) \ge h^{\mathsf{comp}}_{\mathsf{proof}} - (5/m)\log T(n)$;
- column randomness $r_{\mathsf{in}} = r^{\mathsf{out}}_{\mathsf{col}} - r^{\mathsf{in}} \ge w^{\mathsf{out}}_{\mathsf{proof}} - (5/m)\log T(n)$;
- shared randomness $r_{\mathsf{shared}} = r^{\mathsf{out}}_{\mathsf{shared}} + 2\cdot r^{\mathsf{in}} \le (10/m)\log T(n) + O(\log\log T(n)) + O(\log(1/s))$;

We can simply assume that $r^{\mathsf{row}} = h^{\mathsf{comp}}_{\mathsf{proof}} - (5/m)\log T(n)$, $r^{\mathsf{col}} = w^{\mathsf{comp}}_{\mathsf{proof}} - (5/m)\log T(n)$, as we can always move some portion of $\mathsf{seed.row}$ or $\mathsf{seed.col}$ into $\mathsf{seed.shared}$.

Corollary 7.12. *For every constant $\tau < 1$, setting $m := \lceil 20/\tau \rceil$, the above PCPP is τ-almost rectangular with proof length $T(n) \cdot \mathrm{polylog}(T(n)^{1/\tau})$.*

Acknowledgements

Hanlin Ren wants to thank Yuichi Yoshida for helpful discussions about hypergraph cut sparsifiers, Stanislav Živný for helpful discussions about [PZ21], and Lijie Chen for helpful discussions in general and about $\mathsf{THR} \circ \mathsf{THR}$ circuits specifically. We thank Jiatu Li for helpful comments on a draft version of this paper. We thank Jan Pich for helpful discussions during the early stage of this research.

References

[AB18] Amir Abboud and Karl Bringmann. Tighter connections between Formula-SAT and shaving logs. In *Proc. 45th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 8:1–8:18, 2018. doi:10.4230/LIPIcs.ICALP.2018.8. (cit. on p. 32)

[ABRW04] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Pseudorandom generators in propositional proof complexity. *SIAM J. Comput.*, 34(1):67–88, 2004. doi:10.1137/S0097539701389944. (cit. on p. 9, 47)

[AC19] Josh Alman and Lijie Chen. Efficient construction of rigid matrices using an NP oracle. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1034–1055, 2019. doi:10.1109/FOCS.2019.00067. (cit. on p. 3, 11)

[Adl78] Leonard M. Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 75–83, 1978. doi:10.1109/SFCS.1978.37. (cit. on p. 38)

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures Algorithms*, 3(3):289–304, 1992. doi:10.1002/rsa.3240030308. (cit. on p. 79)

[AHM+08] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing disjunctive normal form formulas and AC^0 circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008. doi:10.1137/060664537. (cit. on p. 45)

[AHWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and R. Ryan Williams. Simulating branching programs with edit distance and friends, or: a polylog shaved is a lower bound made. In *Proc. 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 375–388. ACM, 2016. doi:10.1145/2897518.2897653. (cit. on p. 32)

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM Journal of Computing*, 36(4):845–888, 2006. doi:10.1137/S0097539705446950. (cit. on p. 8, 9, 45, 46)

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. doi:10.1145/278298.278306. (cit. on p. 10)

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. doi:10.1145/273865.273901. (cit. on p. 10)

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. doi:10.1016/0022-0000(89)90037-8. (cit. on p. 32)

[BFT98] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *Proc. 13th Annual IEEE Conference on Computational Complexity (CCC)*, pages 8–12, 1998. doi:10.1109/CCC.1998.694585. (cit. on p. 2)

[BGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Short PCPs verifiable in polylogarithmic time. In *Proc. 20th Annual IEEE Conference on Computational Complexity (CCC)*, pages 120–134, 2005. doi:10.1109/CCC.2005.27. (cit. on p. iii, 10, 15, 51, 53, 54, 60)

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. doi:10.1137/S0097539705446810. (cit. on p. 10, 11, 17, 51, 54, 60, 61)

[BHPT20] Amey Bhangale, Prahladh Harsha, Orr Paradise, and Avishay Tal. Rigid matrices from rectangular PCPs or: Hard claims have complex proofs. In *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 858–869, 2020. doi:10.1109/FOCS46700.2020.00084. (cit. on p. 3, 10, 11, 19, 51, 52, 54)

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008. doi:10.1137/050646445. (cit. on p. 11)

[BST19] Nikhil Bansal, Ola Svensson, and Luca Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 910–928. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00059. (cit. on p. 27, 28)

[BSVW03] Eli Ben-Sasson, Madhu Sudan, Salil P. Vadhan, and Avi Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proc. 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 612–621, 2003. doi:10.1145/780542.780631. (cit. on p. 79)

[Bus87] Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 123–131, 1987. doi:10.1145/28395.28409. (cit. on p. 45)

[BV14] Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In *Proc. 41st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 8572 of *Lecture Notes in Computer Science*, pages 163–173, 2014. doi:10.1007/978-3-662-43948-7_14. (cit. on p. 3, 10, 11)

[CGL+19] Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N. Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP = PSPACE. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–20. SIAM, 2019. doi:10.1137/1.9781611975482.1. (cit. on p. 32)

[CH85] Stephen A. Cook and H. James Hoover. A depth-universal circuit. *SIAM J. Comput.*, 14(4):833–839, 1985. doi:10.1137/0214058. (cit. on p. 45)

[Che18] Lijie Chen. Toward super-polynomial size lower bounds for depth-two threshold circuits. *CoRR*, abs/1805.10698, 2018. doi:10.48550/arXiv.1805.10698. (cit. on p. 32)

[Che19] Lijie Chen. Non-deterministic quasi-polynomial time is average-case hard for ACC circuits. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1281–1304, 2019. doi:10.1109/FOCS.2019.00079. (cit. on p. 3)

[CIKK16] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *Proc. 31st Computational Complexity Conference (CCC)*, volume 50 of *LIPIcs*, pages 10:1–10:24, 2016. doi:10.4230/LIPIcs.CCC.2016.10. (cit. on p. 8)

[CKN20] Yu Chen, Sanjeev Khanna, and Ansh Nagda. Near-linear size hypergraph cut sparsifiers. In *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 61–72. IEEE, 2020. doi:10.1109/FOCS46700.2020.00015. (cit. on p. 28)

[CL21] Lijie Chen and Xin Lyu. Inverse-exponential correlation bounds and extremely rigid matrices from a new derandomized XOR lemma. In *Proc. 53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 761–771, 2021. doi:10.1145/3406325.3451132. (cit. on p. 3)

[CLLO21] Lijie Chen, Zhenjian Lu, Xin Lyu, and Igor Carboni Oliveira. Majority vs. approximate linear sum and average-case complexity below NC^1. In *Proc. 48th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 198 of *LIPIcs*, pages 51:1–51:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.51. (cit. on p. 3, 37, 40, 42)

[CLW20] Lijie Chen, Xin Lyu, and R. Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1–12, 2020. doi:10.1109/FOCS46700.2020.00009. (cit. on p. 3, 13, 14, 16, 37, 40, 43, 73)

[COS18] Ruiwen Chen, Igor Carboni Oliveira, and Rahul Santhanam. An average-case lower bound against ACC^0. In *Proc. 13th Latin American Theoretical Informatics Symposium (LATIN)*, volume 10807 of *Lecture Notes in Computer Science*, pages 317–330, 2018. doi:10.1007/978-3-319-77404-6_24. (cit. on p. 3)

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44(1):36–50, 1979. doi:10.2307/2273702. (cit. on p. 47)

[CR20] Lijie Chen and Hanlin Ren. Strong average-case lower bounds from non-trivial derandomization. In *Proc. 52nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1327–1334, 2020. doi:10.1145/3357713.3384279. (cit. on p. 3, 34, 36, 40, 42)

[CW19] Lijie Chen and R. Ryan Williams. Stronger connections between circuit analysis and circuit lower bounds, via PCPs of proximity. In *Proc. 34th Computational Complexity Conference (CCC)*, volume 137 of *LIPIcs*, pages 19:1–19:43, 2019. doi:10.4230/LIPIcs.CCC.2019.19. (cit. on p. 3, 32, 35)

[Din07] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007. doi:10.1145/1236457.1236459. (cit. on p. 10)

[Erd59] Paul Erdős. Graph theory and probability. *Canadian Journal of Mathematics*, 11:34–38, 1959. doi:10.4153/CJM-1959-003-9. (cit. on p. 1)

[FGHK16] Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. A better-than-3n lower bound for the circuit complexity of an explicit function. In *Proc. 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 89–98, 2016. doi:10.1109/FOCS.2016.19. (cit. on p. 1)

[FS16] Lance Fortnow and Rahul Santhanam. New non-uniform lower bounds for uniform classes. In *Proc. 31st Computational Complexity Conference (CCC)*, volume 50 of *LIPIcs*, pages 19:1–19:14, 2016. doi:10.4230/LIPIcs.CCC.2016.19. (cit. on p. 14, 16, 73)

[GR08] Dan Gutfreund and Guy N. Rothblum. The complexity of local list decoding. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*, volume 5171 of *Lecture Notes in Computer Science*, pages 455–468. Springer, 2008. doi:10.1007/978-3-540-85363-3_36. (cit. on p. 4, 37, 38)

[GRST21] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2212–2228, 2021. doi:10.1137/1.9781611976465.132. (cit. on p. 6)

[GS89] Yuri Gurevich and Saharon Shelah. Nearly linear time. In *Logic at Botik '89, Symposium on Logical Foundations of Computer Science, Pereslav-Zalessky, USSR, July 3-8, 1989, Proceedings*, volume 363 of *Lecture Notes in Computer Science*, pages 108–118, 1989. doi:10.1007/3-540-51237-3_10. (cit. on p. 15, 16)

[GSV18] Aryeh Grinberg, Ronen Shaltiel, and Emanuele Viola. Indistinguishability by adaptive procedures with advice, and lower bounds on hardness amplification proofs. In *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 956–966, 2018. doi:10.1109/FOCS.2018.00094. (cit. on p. 4)

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing*, 28(4):1364–1396, 1999. doi:10.1137/S0097539793244708. (cit. on p. 34)

[HMP+93] András Hajnal, Wolfgang Maass, Pavel Pudlák, Mário Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993. doi:10.1016/0022-0000(93)90001-D. (cit. on p. 42)

[HP10] Kristoffer Arnsfelt Hansen and Vladimir V. Podolskii. Exact threshold circuits. In *Proc. 25th Annual IEEE Conference on Computational Complexity (CCC)*, pages 270–279. IEEE Computer Society, 2010. doi:10.1109/CCC.2010.33. (cit. on p. 32, 33)

[HV21] Xuangui Huang and Emanuele Viola. Average-case rigidity lower bounds. In *Proc. 16th International Computer Science Symposium in Russia (CSR)*, volume 12730 of *Lecture Notes in Computer Science*, pages 186–205, 2021. doi:10.1007/978-3-030-79416-3_11. (cit. on p. 3)

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 294–304, 2000. doi:10.1109/SFCS.2000.892118. (cit. on p. 8)

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Proc. 29th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 244–256, 2002. doi:10.1007/3-540-45465-9_22. (cit. on p. 8)

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002. doi:10.1016/S0022-0000(02)00024-7. (cit. on p. 7)

[IM02] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of $5n - o(n)$ for Boolean circuits. In *Proc. 27th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 2420 of *Lecture Notes in Computer Science*, pages 353–364, 2002. doi:10.1007/3-540-45687-2_29. (cit. on p. 1)

[IW01] Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63(4):672–688, 2001. doi:10.1006/jcss.2001.1780. (cit. on p. 7)

[Jeř04] Emil Jeřábek. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Ann. Pure Appl. Log.*, 129(1-3):1–37, 2004. doi:10.1016/j.apal.2003.12.003. (cit. on p. 1)

[KK15] Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proc. 6th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 367–376. ACM, 2015. doi:10.1145/2688073.2688093. (cit. on p. 28)

[KKL+21] Valentine Kabanets, Sajin Koroth, Zhenjian Lu, Dimitrios Myrisiotis, and Igor Carboni Oliveira. Algorithms and lower bounds for De Morgan formulas of low-communication leaf gates. *ACM Trans. Comput. Theory*, 13(4):23:1–23:37, 2021. doi:10.1145/3470861. (cit. on p. 25)

[KKMP21] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou. Total functions in the polynomial hierarchy. In *Proc. 12th Conference on Innovations in Theoretical Computer Science (ITCS)*, volume 185 of *LIPIcs*, pages 44:1–44:18, 2021. doi:10.4230/LIPIcs.ITCS.2021.44. (cit. on p. 1, 2)

[KKTY21] Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In *Proc. 53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 598–611. ACM, 2021. doi:10.1145/3406325.3451061. (cit. on p. 28)

[KL80] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proc. 12th Annual ACM Symposium on Theory of Computing (STOC)*, pages 302–309, 1980. doi:10.1145/800141.804678. (cit. on p. 16)

[Kor21] Oliver Korten. The hardest explicit construction. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 433–444. IEEE, 2021. doi:10.1109/FOCS52979.2021.00051. (cit. on p. 1, 2, 3, 5, 7, 8)

[Kra01] Jan Krajíček. Tautologies from pseudo-random generators. *Bull. Symb. Log.*, 7(2):197–212, 2001. doi:10.2307/2687774. (cit. on p. 1)

[Kra04] Jan Krajíček. Dual weak pigeonhole principle, pseudo-surjective functions, and provability of circuit lower bounds. *J. Symb. Log.*, 69(1):265–286, 2004. doi:10.2178/jsl/1080938841. (cit. on p. 1, 9)

[Kra11] Jan Krajíček. On the proof complexity of the Nisan-Wigderson generator based on a hard $\mathsf{NP} \cap \mathsf{coNP}$ function. *J. Math. Log.*, 11(1), 2011. doi:10.1142/S0219061311000979. (cit. on p. 9)

[KW98] Johannes Köbler and Osamu Watanabe. New collapse consequences of NP having small circuits. *SIAM J. Comput.*, 28(1):311–324, 1998. doi:10.1137/S0097539795296206. (cit. on p. 2)

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Comb.*, 7(4):357–363, 1987. doi:10.1007/BF02579323. (cit. on p. 37)

[LW17] Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2182–2189. SIAM, 2017. doi:10.1137/1.9781611974782.142. (cit. on p. 6)

[LY22] Jiatu Li and Tianqi Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. In *Proc. 54th Annual ACM Symposium on Theory of Computing (STOC)*, 2022. To appear. URL: https://eccc.weizmann.ac.il/report/2021/023/. (cit. on p. 1)

[Mie09] Thilo Mie. Short PCPPs verifiable in polylogarithmic time with $O(1)$ queries. *Ann. Math. Artif. Intell.*, 56(3-4):313–338, 2009. doi:10.1007/s10472-009-9169-y. (cit. on p. 21, 51, 65)

[Mur71] Saburo Muroga. *Threshold logic and its applications*. Wiley, 1971. (cit. on p. 33)

[MW20] Cody D. Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM J. Comput.*, 49(5), 2020. doi:10.1137/18M1195887. (cit. on p. 3)

[Nep70] V. A. Nepomnjascii. Rudimentary predicates and Turing computations. In *Doklady Akademii Nauk*, volume 195, pages 282–284. Russian Academy of Sciences, 1970. (cit. on p. 8, 45)

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1. (cit. on p. 34, 38, 42, 43)

[OS17] Igor Carboni Oliveira and Rahul Santhanam. Conspiracies between learning algorithms, circuit lower bounds, and pseudorandomness. In *Proc. 32nd Computational Complexity Conference (CCC)*, volume 79 of *LIPIcs*, pages 18:1–18:49, 2017. doi:10.4230/LIPIcs.CCC.2017.18. (cit. on p. 8)

[Pin73] Mark S Pinsker. On the complexity of a concentrator. In *7th International Telegraffic Conference*, volume 4, pages 1–318, 1973. (cit. on p. 1)

[PS94] Ramamohan Paturi and Michael E. Saks. Approximating threshold circuits by rational functions. *Inf. Comput.*, 112(2):257–272, 1994. doi:10.1006/inco.1994.1059. (cit. on p. 33)

[PT07] Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proc. 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–271, 2007. doi:10.1109/FOCS.2007.54. (cit. on p. 6)

[PZ21] Eden Pelleg and Stanislav Zivný. Additive sparsification of CSPs. In *Proc. 29th European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 75:1–75:15, 2021. doi:10.4230/LIPIcs.ESA.2021.75. (cit. on p. 27)

[Raz15] Alexander Razborov. Pseudorandom generators hard for k-DNF resolution and polynomial calculus resolution. *Annals of Mathematics*, 181(2):415–472, 2015. doi:10.4007/annals.2015.181.2.1. (cit. on p. 9)

[Rei11] Ben Reichardt. Reflections for quantum query algorithms. In *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 560–569. SIAM, 2011. doi:10.1137/1.9781611973082.44. (cit. on p. 25)

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997. doi:10.1006/jcss.1997.1494. (cit. on p. 8)

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell System technical journal*, 28(1):59–98, 1949. doi:10.1002/j.1538-7305.1949.tb03624.x. (cit. on p. 1)

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996. doi:10.1109/18.556668. (cit. on p. 16)

[SV10] Ronen Shaltiel and Emanuele Viola. Hardness amplification proofs require majority. *SIAM J. Comput.*, 39(7):3122–3154, 2010. doi:10.1137/080735096. (cit. on p. 4)

[SW13] Rahul Santhanam and R. Ryan Williams. On medium-uniformity and circuit lower bounds. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 15–23. IEEE Computer Society, 2013. doi:10.1109/CCC.2013.40. (cit. on p. 3)

[SY19] Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2570–2581. SIAM, 2019. doi:10.1137/1.9781611975482.159. (cit. on p. 28)

[Tal17] Avishay Tal. Formula lower bounds via the quantum method. In *Proc. 49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1256–1268. ACM, 2017. doi:10.1145/3055399.3055472. (cit. on p. 6, 25)

[Vad12] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012. doi:10.1561/0400000010. (cit. on p. 60)

[Val77] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *Proc. 6th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176, 1977. doi:10.1007/3-540-08353-7_135. (cit. on p. 1)

[Vio20] Emanuele Viola. New lower bounds for probabilistic degree and AC^0 with parity gates. *Electron. Colloquium Comput. Complex.*, page 15, 2020. URL: https://eccc.weizmann.ac.il/report/2020/015. (cit. on p. 3)

[VW18] Emanuele Viola and Avi Wigderson. Local expanders. *Computational Complexity*, 27(2):225–244, 2018. doi:10.1007/s00037-017-0155-1. (cit. on p. 58)

[VW20] Nikhil Vyas and R. Ryan Williams. Lower bounds against sparse symmetric functions of ACC circuits: Expanding the reach of #SAT algorithms. In *Proc. 37th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154 of *LIPIcs*, pages 59:1–59:17, 2020. doi:10.4230/LIPIcs.STACS.2020.59. (cit. on p. 3)

[Wil13] R. Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal of Computing*, 42(3):1218–1244, 2013. doi:10.1137/10080703X. (cit. on p. 3, 8, 32)

[Wil14] R. Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014. doi:10.1145/2559903. (cit. on p. 3)

[Wil16] R. Ryan Williams. Natural proofs versus derandomization. *SIAM Journal of Computing*, 45(2):497–529, 2016. doi:10.1137/130938219. (cit. on p. 3, 7)

[Wil18a] R. Ryan Williams. Limits on representing Boolean functions by linear combinations of simple functions: Thresholds, ReLUs, and low-degree polynomials. In *Proc. 33rd Computational Complexity Conference (CCC)*, volume 102 of *LIPIcs*, pages 6:1–6:24, 2018. doi:10.4230/LIPIcs.CCC.2018.6. (cit. on p. 3)

[Wil18b] R. Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. *Theory Comput.*, 14(1):1–25, 2018. doi:10.4086/toc.2018.v014a017. (cit. on p. 3)

[Zák83] Stanislav Zák. A Turing machine time hierarchy. *Theor. Comput. Sci.*, 26:327–333, 1983. doi:10.1016/0304-3975(83)90015-4. (cit. on p. 11, 13)

A Omitted Proofs

A.1 Proof of Theorem 2.3

We need the following binary search algorithm.

Lemma A.1 ([CLW20, Lemma 4.4]). *There is an algorithm A satisfying the following.*

- **Input.** *A is given an explicit integer $n \geq 2$ (written in binary form) as input, together with oracle access to a list $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ such that $a_1 \neq a_n$.*
- **Output.** *An index $p \in [1, n-1]$ such that $a_p \neq a_{p+1}$.*
- **Efficiency.** *A runs in $O(\log n)$ time and makes at most $O(\log n)$ queries to the list.*

Theorem 2.3 ([FS16, CLW20]). *Let c be a large universal constant, $T : \mathbb{N} \to \mathbb{N}$ be a good function such that $n \log^{c+1} n \leq T(n) \leq 2^{\mathrm{poly}(n)}$. There is a language*

$$L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathrm{TM}}[T(n)] \setminus \text{i.o.-}\mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T(n)/\log^c T(n), n/10]_{/(n/10)}.$$

Moreover, there is an algorithm $\mathcal{R}$ (the "refuter") such that the following holds.

(Input) *$\mathcal{R}$ receives three inputs $(1^n, M, \alpha)$, where M is a nondeterministic RTM and $\alpha \in \{0,1\}^{n/10}$ is an advice string. It is guaranteed that M runs in $T(n)/\log^c T(n)$ time and uses at most $n/10$ nondeterministic bits; moreover, the description length of M is $O(1)$.*

(Output) *For every fixed M, every sufficiently large n, and every advice $\alpha \in \{0,1\}^{n/10}$, $\mathcal{R}(1^n, M, \alpha)$ outputs a string $x \in \{0,1\}^n$ such that $M(x; \alpha) \neq L^{\mathsf{hard}}(x)$.*

(Complexity) *$\mathcal{R}$ runs in $\mathrm{poly}(T(n))$ time with adaptive access to an NP oracle.*

Proof. We first define L^{hard}. Let $x \in \{0,1\}^n$ be the input of L^{hard}, we parse it into $x := (\langle M \rangle, \alpha, w, x_{\mathsf{rest}})$. Here, $\langle M \rangle$ is the description of a nondeterministic RTM M, $\alpha \in \{0,1\}^{n/10}$ is the advice string for M, $w \in \{0,1\}^{n/10}$ is a witness of M, and x_{rest} denotes the rest input bits. We interpret M as a nondeterministic RTM that guesses at most $n/10$ nondeterministic bits and runs in at most $T' := T/\log^c T$ time; if the nondeterminism or time complexity exceeds the corresponding bounds, we force M to reject.

(i) If M accepts the input $(\langle M \rangle, \alpha, 0^{n/10}, x_{\mathsf{rest}})$ with witness w and advice α, then $L^{\mathsf{hard}}(x) = 0$.

(ii) Otherwise, if $w = 1^{n/10}$, then $L^{\mathsf{hard}}(x) = 1$.

(iii) Otherwise, let $w+1$ be the lexicographically next string after w, $L^{\mathsf{hard}}(x) = 1$ if and only if M accepts $(\langle M \rangle, \alpha, w+1, x_{\mathsf{rest}})$ with advice α.

Since a nondeterministic RTM of time complexity $T' = T/\log^c T$ can be simulated by a nondeterministic TM of time complexity $o(T)$, it follows that $L^{\mathsf{hard}} \in \mathsf{NTIME}_{\mathrm{TM}}[T(n)]$.

Before describing the refuter, it is instructive to understand why L^{hard} does not admit a nondeterministic RTM algorithm with time T', $n/10$ nondeterministic bits, and $n/10$ advice bits. Let M be such an algorithm and $\alpha \in \{0,1\}^{n/10}$ be the corresponding advice string. For the sake of contradiction, suppose M computes L^{hard}. Fix an arbitrary x_{rest}. For a witness string $w \in \{0,1\}^{n/10}$, denote $x_w := (\langle M \rangle, \alpha, w, x_{\mathsf{rest}})$. Abusing notation, for an integer $0 \le i < 2^{n/10}$, let w_i be the $(i+1)$-th lexicographically smallest string (with $w_0 = 0^{n/10}$ and $w_{2^{n/10}-1} = 1^{n/10}$), we also denote $x_i := x_{w_i}$.

- Suppose $M(x_0)$ accepts. Let w be the lexicographically smallest witness w such that M accepts x_0 on witness string w. It follows from (i) that $L^{\mathsf{hard}}(x_w) = 0$. However, for every $w' < w$, since M does not accept $x_{w'}$, by (iii) we have that

  $$L^{\mathsf{hard}}(x_{w'}) = M(x_{w'+1}) = L^{\mathsf{hard}}(x_{w'+1}).$$

  It follows that $1 = M(x_0) = L^{\mathsf{hard}}(x_0) = L^{\mathsf{hard}}(x_w) = 0$, a contradiction.

- Suppose $M(x_0)$ rejects. Then M rejects x_0 on every possible witness, which means (i) never happens. It follows from (iii) that for every $w \in \{0,1\}^{n/10} \setminus \{1^{n/10}\}$,

  $$L^{\mathsf{hard}}(x_w) = M(x_{w+1}) = L^{\mathsf{hard}}(x_{w+1}).$$

  This is a contradiction as $L^{\mathsf{hard}}(x_0) = 0$ but $L^{\mathsf{hard}}(x_{1^{n/10}}) = 1$.

Now we describe our refuter. On input $(1^n, M, \alpha)$, our refuter $\mathcal{R}$ first uses the NP oracle to decide if $M(x_0)$ accepts.

- If $M(x_0)$ accepts, then it uses the NP oracle to find the lexicographically smallest w such that M accepts x_0 on witness string w. Consider the list

  $$1 = M(x_0), M(x_1), \ldots, M(x_w), L^{\mathsf{hard}}(x_w) = 0.$$

  We use Lemma A.1 to find two adjacent entries in the list that are different. This takes $O(\log(2^n)) = O(n)$ time with random access to the list. As random access to the list can be simulated by an NP oracle, this pair of entries can be found in polynomial time with an NP oracle. Now there are two cases:

– (Case I) Suppose the two entries are $M(x_w)$ and $L^{\mathsf{hard}}(x_w)$. The refuter simply outputs x_w.
– (Case II) Suppose the two entries are $M(x_{w'})$ and $M(x_{w'+1})$ where $w' < w$. Since M does not accept x_0 on witness string w' and $w' < 1^{n/10}$, (iii) applies to w'. Therefore $L^{\mathsf{hard}}(x_{w'}) = M(x_{w'+1}) \neq M(x_{w'})$, and the refuter can output $x_{w'}$.

- If $M(x_0)$ rejects, then consider the list

$$0 = M(x_0), M(x_1), \ldots, M(x_{1^{n/10}}), L^{\mathsf{hard}}(x_{1^{n/10}}) = 1.$$

Again, we use Lemma A.1 to find two adjacent entries in the list that are different, in polynomial time with an NP oracle. There are two cases:

– (Case I) Suppose the two entries are $M(x_{1^{n/10}})$ and $L^{\mathsf{hard}}(x_{1^{n/10}})$. The refuter simply outputs $x_{1^{n/10}}$.
– (Case II) Suppose the two entries are $M(x_w)$ and $M(x_{w+1})$ for some $w \in \{0,1\}^{n/10} \setminus \{1^{n/10}\}$. Since M does not accept x_0 (at all) and $w < 1^{n/10}$, (iii) applies to w. Therefore $L^{\mathsf{hard}}(x_w) = M(x_{w+1}) \neq M(x_w)$, and the refuter can output x_w. □

A.2 Proof of Lemma 4.10

Lemma 4.10. *Let $\epsilon, \delta > 0$, $\mathscr{C}$ be a circuit class, $s(n)$ be a good function, and $G : \{0,1\}^r \to \{0,1\}^n$ be a PRG that ϵ-fools $\mathscr{C}$ circuits of size $s(n)$. For $\epsilon' := 2\delta + \epsilon \cdot s(n)$, G also ϵ'-fools $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuits of complexity $s(n)$.*

Proof. Let C be a $\widetilde{\mathsf{Sum}}_\delta \circ \mathscr{C}$ circuit where the underlying (real-valued) $\mathsf{Sum} \circ \mathscr{C}$ circuit is

$$\tilde{C} := \sum_{i=1}^{\ell} \alpha_i \cdot C_i.$$

Let $\mathcal{U}_n$ denote the uniform distribution over $\{0,1\}^n$; we abuse notation and let G_n denote the distribution of $G_n(y)$ for a uniformly random $y \in \{0,1\}^r$. From the definition of $\widetilde{\mathsf{Sum}}_\delta$ gates, we have

$$\left|\mathbb{E}[C(\mathcal{U}_n)] - \mathbb{E}[\tilde{C}(\mathcal{U}_n)]\right| \leq \delta \quad \text{and} \quad \left|\mathbb{E}[C(G_n)] - \mathbb{E}[\tilde{C}(G_n)]\right| \leq \delta.$$

Using the property of G_n we have

$$\begin{aligned}
\left|\mathbb{E}[\tilde{C}(\mathcal{U}_n)] - \mathbb{E}[\tilde{C}(G_n)]\right| &= \left|\mathbb{E}\left[\sum_{i=1}^{\ell} \alpha_i \cdot C_i(\mathcal{U}_n)\right] - \mathbb{E}\left[\sum_{i=1}^{\ell} \alpha_i \cdot C_i(G_n)\right]\right| \\
&= \left|\sum_{i=1}^{\ell} \alpha_i \cdot (\mathbb{E}[C_i(\mathcal{U}_n)] - \mathbb{E}[C_i(G_n)])\right| \\
&\leq \left(\max_{i=1}^{\ell} |\mathbb{E}[C_i(\mathcal{U}_n)] - \mathbb{E}[C_i(G_n)]|\right) \cdot \left(\sum_{i=1}^{\ell} |\alpha_i|\right) \\
&\leq \epsilon \cdot s(n).
\end{aligned}$$

Therefore,

$$\begin{aligned}|\mathbb{E}[C(\mathcal{U}_n)] - \mathbb{E}[C(G_n)]| &\le \left|\mathbb{E}[C(\mathcal{U}_n)] - \mathbb{E}[\tilde{C}(\mathcal{U}_n)]\right| + \left|\mathbb{E}[\tilde{C}(\mathcal{U}_n)] - \mathbb{E}[\tilde{C}(G_n)]\right| \\ &\quad + \left|\mathbb{E}[\tilde{C}(G_n)] - \mathbb{E}[C(G_n)]\right| \\ &\le 2\delta + \epsilon \cdot s(n) = \epsilon'.\end{aligned}$$

□

A.3 Proof of Lemma 4.11

Lemma 4.11. *Let $d \ge 2$ be a constant, $s = s(n), T = T(n)$ be good functions, and $\mathscr{C}$ be a circuit class. Suppose that there is a deterministic $T(n)$-time algorithm for the* CAPP *problem for $\mathsf{AND}_d \circ \mathscr{C}$ circuits with inverse-circuit-size error, where the bottom $\mathscr{C}$ circuits have size $\Theta(s(n)^d)$.*

Then, there is a constant $\delta > 0$ and an algorithm that achieves the following.

(Input) *The algorithm is given d $\mathsf{Sum} \circ \mathscr{C}$ circuits $C_1, C_2, \ldots, C_d$ as inputs, where for each $i \in [d]$, C_i has complexity at most $s(n)$, and it is promised that $\|C_i - \mathsf{bin}_{C_i}\|_d \le \delta$.*

(Output) *The algorithm estimates the following quantity within an additive error of $(1/6) \cdot 2^{-d}$.*

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[\mathsf{bin}_{C_1}(x) \wedge \mathsf{bin}_{C_2}(x) \wedge \cdots \wedge \mathsf{bin}_{C_d}(x)].$$

(Complexity) *The algorithm runs in deterministic $O(s(n)^d \cdot T(n))$ time.*

Proof. Suppose we are given an $\mathsf{AND}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit as input:

$$C(x) := \mathsf{bin}_{C_1}(x) \wedge \mathsf{bin}_{C_2}(x) \wedge \cdots \wedge \mathsf{bin}_{C_d}(x),$$

where for each $i \in [d]$, C_i is a $\mathsf{Sum} \circ \mathscr{C}$ circuit with $\|C_i - \mathsf{bin}_{C_i}\|_d \le \delta$.

Now we want to estimate

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}\left[\prod_{i=1}^{d} \mathsf{bin}_{C_i}(x)\right]. \tag{9}$$

First we show that

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}\left[\prod_{i=1}^{d} C_i(x)\right] \tag{10}$$

is a good estimation for Eq. (9).

Recall that for every $1 \le i \le d$, we have $\|C_i - \mathsf{bin}_{C_i}\|_d \le \delta$. As bin_{C_i} are Boolean functions, $\|\mathsf{bin}_{C_i}\|_d \le 1$. Using the triangular inequality we have $\|C_i\|_d \le 1 + \delta$. By Lemma 4.9 we have

$$|(9) - (10)| = |\langle C_1, \ldots, C_d\rangle - \langle \mathsf{bin}_{C_1}, \ldots, \mathsf{bin}_{C_d}\rangle| \le d \cdot (1+\delta)^{d-1} \cdot \delta.$$

Thus for any constant d, we can choose a suitable constant δ such that $|(9) - (10)| \le 1/12 \cdot 2^{-d}$.

Now we show how to compute a good approximation for Eq. (10). Let the linear combinations of each C_i circuit be $\sum_{j \in S_i} \alpha_{i,j} C_{i,j}$, then

$$\begin{aligned}(10) &= \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}\left[\prod_{i=1}^{d}\left(\sum_{j \in S_i} \alpha_{i,j} C_{i,j}(x)\right)\right] \\ &= \sum_{p \in S_1 \times S_2 \times \ldots \times S_d} \alpha_{1,p_1}\alpha_{2,p_2}\cdots\alpha_{d,p_d} \cdot \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[C_{1,p_1}(x)C_{2,p_2}(x)\cdots C_{d,p_d}(x)]\end{aligned}$$

Since the product of d $\mathscr{C}$ circuits is a $\mathsf{AND}_d \circ \mathscr{C}$ circuit, we can use the CAPP algorithm for $\mathsf{AND}_d \circ \mathscr{C}$ to estimate the following quantity within additive error $2^{-d}/(12s(n)^d)$.

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[C_{1,p_1}(x)C_{2,p_2}(x)\dots C_{d,p_d}(x)]$$

Thus we can estimate Eq. (10) with error

$$\begin{aligned}
&(2^{-d}/(12s(n)^d)) \cdot \left(\sum_{p \in S_1 \times S_2 \times \ldots \times S_d} |\alpha_{1,p_1}| \cdot |\alpha_{2,p_2}| \cdot \ldots \cdot |\alpha_{d,p_d}| \right) \\
&\leq (2^{-d}/(12s(n)^d)) \cdot \left(\sum_{i \in S_1} |\alpha_{1,i}| \right) \cdot \left(\sum_{i \in S_2} |\alpha_{2,i}| \right) \cdot \ldots \cdot \left(\sum_{i \in S_d} |\alpha_{d,i}| \right) \\
&\leq (2^{-d}/(12s(n)^d)) \cdot s(n)^d \\
&\leq 1/12 \cdot 2^{-d}.
\end{aligned}$$

It follows that we could solve the CAPP problem for such circuits with error $1/12 \cdot 2^{-d} + 1/12 \cdot 2^{-d} \leq 1/6 \cdot 2^{-d}$. The algorithm runs in deterministic $O(s(n)^d \cdot T(n))$ time. □

A.4 Proof of Lemma 4.12

Lemma 4.12. *Let $S \in \mathbb{N}$ and $d \geq 2$ where d is an even number. Suppose we are given S reals $(\alpha_i)_{i \in [S]}$, S $\mathscr{C}$ circuits $(C_i)_{i \in [S]}$, and a parameter $\delta < 0.01/d$. Let $\alpha_{\mathsf{tot}} := \sum_{i \in [S]} |\alpha_i|$ and $\epsilon := \frac{\delta^d}{2 \cdot 3^d \cdot (\alpha_{\mathsf{tot}}+1)^{2d}}$. Suppose the CAPP problem for $\mathsf{AND}_{2d} \circ \mathscr{C}$ with error ϵ can be solved in $T(n)$ time. Let $f = \sum_{i=1}^{S} \alpha_i \cdot C_i$. There is an algorithm A running in $O(T(n) \cdot (S+1)^{2d})$ time such that:*

- *If one of the following conditions hold, then A always accepts;*
  - $\|f - \mathsf{bin}_f\|_\infty \leq \delta/3$
  - $\|f - \mathsf{bin}_f\|_1 \leq (\delta/3)^d$ *and* $f(x) \in [0,1]$ *for any* $x \in \{0,1\}^n$
- *if $\|f - \mathsf{bin}_f\|_d \geq \delta$, then A always rejects;*
- *otherwise, A can output anything.*

Proof. We define a polynomial $P(z) = z^d(1-z)^d$. We also define $d_{\mathsf{bin}}(z)$ to be $\min(|z|, |z-1|)$. Simple calculations confirm the following facts about $P(z)$:

- $P(z) \leq d_{\mathsf{bin}}(z)^d \cdot (1 + d_{\mathsf{bin}}(z))^d$, and
- $P(z) \geq d_{\mathsf{bin}}(z)^d \cdot 2^{-d}$.

We now show the following properties of p:

1. When $d_{\mathsf{bin}}(z) \leq \delta/3$, we have $P(z) \leq (\delta/3)^d \cdot (1+\delta/3)^d$. This means that if $\|f - \mathsf{bin}_f\|_\infty \leq \delta/3$, then we have

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[P(f(x))] \leq (\delta/3)^d \cdot (1+\delta/3)^d \leq (\delta/3)^d \cdot (1+0.01/d)^d \leq (\delta/3)^d \cdot e^{0.01}.$$

2. When $f(x) \in [0,1]$ for any $x \in \{0,1\}^n$, we have for every $x \in \{0,1\}^n$, $P(x) < d_{\mathsf{bin}}(x)$ and $\mathbb{E}_{x \leftarrow \{0,1\}^n}[P(f(x))] \leq \|f - \mathsf{bin}_f\|_1$. Thus when $\|f - \mathsf{bin}_f\|_1 \leq (\delta/3)^d$ also holds, we have $\mathbb{E}_{x \leftarrow \{0,1\}^n}[P(f(x))] \leq (\delta/3)^d$.

3. If $\|f - \mathsf{bin}_f\|_d \geq \delta$, then by definition we have

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}\left[d_{\mathsf{bin}}(f(x))^d\right] \geq \delta^d.$$

Since $d \geq 2$, we have

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[P(f(x))] \geq (\delta/2)^d \geq (9/4) \cdot (\delta/3)^d.$$

Therefore, it suffices to compute

$$\mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[P(f(x))] \tag{11}$$

within error $(\delta/3)^d/2$ to distinguish between these two cases in the lemma.

Expanding out $P(f(x)) = P\left(\sum_{i=1}^{S} \alpha_i \cdot C_i\right)$, it can be written as an $\mathbb{R}$-sum of at most $(S+1)^{2d}$ products of $2d$ functions from $\mathscr{C}$. By rearranging the order of summation (summing all $(S+1)^{2d}$ terms first), we see that (11) can be evaluated by making at most $(S+1)^{2d}$ calls to the assumed algorithm within error $\frac{\delta^d}{2 \cdot 3^d \cdot (\alpha_{\mathrm{tot}}+1)^{2d}}$. Assuming that algorithm runs in $T(n)$ time, the sum (11) can be evaluated in time $O(T(n) \cdot (S+1)^{2d})$. □

A.5 Proof of Lemma 4.14

Lemma 4.14. *Let $s = s(n)$ be a good function, $G : \{0,1\}^{n-1} \to \{0,1\}^n$ be a seed-extending PRG that ϵ-fools $\mathscr{C}$ circuits of size $s(n)$. Then the following problem L has ℓ_1-distance at least $(1/2 - \epsilon \cdot s(n))$ from $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuits of complexity $s(n)$.*

$$L = \left\{y \in \{0,1\}^n : \exists x \in \{0,1\}^{n-1}, \text{s.t. } G(x) = y\right\}.$$

Proof. Fix any $[0,1]$-$\mathsf{Sum} \circ \mathscr{C}$ circuit C of complexity s. That is:

$$C(x) = \sum_{i=1}^{\ell} \alpha_i C_i(x),$$

where $\sum_{i=1}^{\ell} |\alpha_i| \leq s$ and $\sum_{i=1}^{\ell} |C_i| \leq s$. It follows that

$$\begin{aligned}
\|L - C\|_1 &= \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[|L(x) - C(x)|] \\
&= \frac{1}{2^n} \cdot \left(\sum_{x \in \mathrm{Range}(G)} (1 - C(x)) + \sum_{x \in \{0,1\}^n \setminus \mathrm{Range}(G)} C(x) \right) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left(\mathop{\mathbb{E}}_{x \in \{0,1\}^n \setminus \mathrm{Range}(G)}[C(x)] - \mathop{\mathbb{E}}_{x \in \mathrm{Range}(G)}[C(x)] \right). \qquad (12)
\end{aligned}$$

Here, Eq. (12) is because G is seed-extending and $|\mathrm{Range}(G)| = 2^{n-1}$.

Let $E_{\mathsf{in}} := \mathbb{E}_{x \in \mathrm{Range}(G)}[C(x)]$ and $E_{\mathsf{out}} := \mathbb{E}_{x \in \{0,1\}^n \setminus \mathrm{Range}(G)}[C(x)]$. Then $\mathbb{E}_{x \leftarrow \{0,1\}^n}[C(x)] = (E_{\mathsf{in}} + E_{\mathsf{out}})/2$. By the same argument as in the proof of Lemma 4.10, we have

$$|E_{\mathsf{in}} - E_{\mathsf{out}}|/2 = \left| \mathop{\mathbb{E}}_{x \leftarrow \{0,1\}^n}[C(x)] - E_{\mathsf{in}} \right| \leq \epsilon \cdot s(n).$$

It follows that

$$(12) = (1 + E_{\mathsf{in}} - E_{\mathsf{out}})/2 \geq 1/2 - \epsilon \cdot s(n). \qquad \square$$

A.6 Proof of Lemma 7.2

For completeness, we include the definition of λ-biased sets:

Definition A.2. For two strings $x, y \in \{0,1\}^n$, denote their inner product as $\mathrm{IP}(x,y) := (-1)^{\sum_{i=1}^n x_i y_i}$.

Let $\lambda > 0$. A set $S \subseteq \{0,1\}^n$ is a *λ-biased set* if for every string $y \in \{0,1\}^n$, if y is not the all-zero string, then

$$\left| \sum_{x \in S} \mathrm{IP}(x,y) \right| \leq \lambda |S|$$

Note that when $\mathbb{F} = \mathrm{GF}(2^q)$, the definition of λ-biased subsets of $\mathbb{F}^m$ in [BSVW03, Section 2] coincides with the above definition (of λ-biased subsets of $\{0,1\}^{mq}$).

We use the explicit construction of a λ-biased subset $S_\lambda \subseteq \mathbb{F}^m$ of size $O((qm/\lambda)^2)$ in [AGHP92]. Recall that we also need that the first coordinate of every element in S_λ is nonzero. Therefore, we first construct a $(\lambda/4)$-biased set $S_{\lambda/4} \subseteq \mathbb{F}^m$ using [AGHP92], and then remove every element $\vec{y} \in S_{\lambda/4}$ with $y_1 = 0$ to obtain the λ-biased set S_λ. In what follows, we prove that this remaining set is indeed a λ-biased set.

Lemma 7.2. *Let $\lambda < 0.1$, q, m be integers such that $q \geq \log \frac{4}{\lambda}$, and let $\mathbb{F} = \mathrm{GF}(2^q)$. There is a deterministic polynomial-time algorithm that on input $(1^m, 1^q, 1^{\lceil 1/\lambda \rceil})$, outputs a λ-biased set $S_\lambda \subseteq (\mathbb{F} \setminus \{0\}) \times \mathbb{F}^{m-1}$ of size $O((qm/\lambda)^2)$.*

Proof. We first use [AGHP92] to construct a $\lambda/4$-biased set $S_{\lambda/4} \subseteq \mathbb{F}^m$ of size $O((qm/\lambda)^2)$. Let $S_0 := \{y \in S_{\lambda/4} : y_1 = 0\}$ and $S_\lambda := S_{\lambda/4} \setminus S_0$, we will show that S_λ is a λ-biased set.

First, we show that $|S_0| \leq (\lambda/2)|S_{\lambda/4}|$, i.e., we only removed a small fraction of elements. Let $\mathcal{Y} := \{0,1\}^q \times \{0^{(m-1)q}\}$ be the set of length-(mq) strings that is zero on all but the first q input bits; each $y \in \mathcal{Y}$ corresponds to a linear test $\mathrm{IP}(\cdot, y)$ that *only depends on the first q input bits.* Abusing notation, we also use $\mathcal{Y}$ to denote the uniform distribution over $\mathcal{Y}$. For every $x \in S_{\lambda/4}$, if $x \in S_0$ then $\mathrm{IP}(x,y) = 1$ for every $y \in \mathcal{Y}$; otherwise the expectation of $\mathrm{IP}(x,y)$ over a random $y \leftarrow \mathcal{Y}$ is zero. It follows that

$$|S_0|/|S_{\lambda/4}| = \mathop{\mathbb{E}}_{x \leftarrow S_{\lambda/4}, y \leftarrow \mathcal{Y}}[\mathrm{IP}(x,y)].$$

On the other hand, if $y = 0^{mq}$ then $\mathrm{IP}(x,y) = 1$ for every possible x, while if $y \neq 0^{mq}$ then the expectation of $\mathrm{IP}(x,y)$ over a random $x \leftarrow S_{\lambda/4}$ is between $-\lambda/4$ and $\lambda/4$. Therefore

$$\mathop{\mathbb{E}}_{x \leftarrow S_{\lambda/4}, y \leftarrow \mathcal{Y}}[\mathrm{IP}(x,y)] \leq \frac{1}{2^q} + \lambda/4 \leq \lambda/2.$$

Now we show that S_λ is a λ-biased set. Fix any binary string $y \in \{0,1\}^{mq} \setminus \{0^{mq}\}$, we have

$$\begin{aligned}\left|\sum_{x\in S_\lambda} \mathrm{IP}(x,y)\right| &\le \left|\sum_{x\in S_{\lambda/4}} \mathrm{IP}(x,y)\right| + \left|\sum_{x\in S_0} \mathrm{IP}(x,y)\right| \\ &\le (\lambda/4)|S_{\lambda/4}| + |S_0| \\ &\le (3\lambda/4)|S_{\lambda/4}|.\end{aligned}$$

On the other hand, we have $|S_\lambda| \ge (1-\lambda/2)|S_{\lambda/4}| \ge 0.95|S_{\lambda/4}|$. Therefore

$$\left|\mathop{\mathbb{E}}_{x\leftarrow S_\lambda}[\mathrm{IP}(x,y)]\right| \le \frac{3\lambda}{4\cdot 0.95} \le \lambda. \qquad \square$$

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il