Electronic Colloquium on Computational Complexity, Report No. 72 (2023)

Range Avoidance, Remote Point, and Hard Partial Truth Tables via Satisfying-Pairs Algorithms

Yeyuan Chen
Xi'an Jiaotong University
yychen9961@gmail.com

Yizhi Huang
IIIS, Tsinghua University
huangyizhi01@gmail.com

Jiatu Li
IIIS, Tsinghua University
lijt19@mails.tsinghua.edu.cn

Hanlin Ren
University of Oxford
h4n1in.r3n@gmail.com

May 18, 2023

Abstract

The *range avoidance problem*, denoted as $\mathscr{C}$-Avoid, asks to find a non-output of a given $\mathscr{C}$-circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ with stretch $\ell > n$. This problem has recently received much attention in complexity theory for its connections with circuit lower bounds and other explicit construction problems. Inspired by the Algorithmic Method for circuit lower bounds, Ren, Santhanam, and Wang (FOCS'22) established a framework to design $\mathsf{FP}^{\mathsf{NP}}$ algorithms for $\mathscr{C}$-Avoid via *slightly non-trivial* data structures related to $\mathscr{C}$. However, a major drawback of their approach is the lack of unconditional results even for $\mathscr{C} = \mathsf{AC}^0$.

In this work, we present the first unconditional $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Avoid. Indeed, we obtain $\mathsf{FP}^{\mathsf{NP}}$ algorithms for the following stronger problems:

- **(ACC^0-Remote-Point).** Given $C : \{0,1\}^n \to \{0,1\}^\ell$ for some $\ell = \text{quasi-poly}(n)$ such that each output bit of C is computed by a quasi-poly(n)-size $\mathsf{AC}^0[m]$ circuit, we can find some $y \in \{0,1\}^\ell$ in $\mathsf{FP}^{\mathsf{NP}}$ such that for every $x \in \{0,1\}^n$, the relative Hamming distance between y and $C(x)$ is at least $1/2 - 1/\text{poly}(n)$. This problem is the "average-case" analogue of ACC^0-Avoid.
- **(ACC^0-Partial-AvgHard).** Given $x_1, \ldots, x_\ell \in \{0,1\}^n$ for some $\ell = \text{quasi-poly}(n)$, we can compute ℓ bits $y_1, \ldots, y_\ell \in \{0,1\}$ in $\mathsf{FP}^{\mathsf{NP}}$ such that for every $2^{\log^c(n)}$-size ACC^0 circuit C, $\Pr_i[C(x_i) \neq y_i] \geq 1/2 - 1/\text{poly}(n)$, where $c = O(1)$. This problem generalises the strong average-case circuit lower bounds against ACC^0 in a different way.

Our algorithms can be seen as natural generalisations of the best known almost-everywhere average-case lower bounds against ACC^0 circuits by Chen, Lyu, and Williams (FOCS'20). Note that both problems above have been studied prior to our work, and no $\mathsf{FP}^{\mathsf{NP}}$ algorithm was known even for weak circuit classes such as GF(2)-linear circuits and DNF formulas.

Our results follow from a strengthened algorithmic method: slightly non-trivial algorithms for the *Satisfying-Pairs* problem for $\mathscr{C}$ implies $\mathsf{FP}^{\mathsf{NP}}$ algorithms for $\mathscr{C}$-Avoid (as well as $\mathscr{C}$-Remote-Point and $\mathscr{C}$-Partial-AvgHard). Here, given $\mathscr{C}$-circuits $\{C_i\}$ and inputs $\{x_j\}$, the $\mathscr{C}$-Satisfying-Pairs problem asks to (approximately) count the number of pairs (i, j) such that $C_i(x_j) = 1$.

A technical contribution of this work is a construction of a *short, smooth, and rectangular PCP of Proximity* that combines two previous PCP constructions, which may be of independent interest. It serves as a key tool that allows us to generalise the framework for Avoid to the average-case scenarios.

ISSN 1433-8092

Contents

1 Introduction

> "You might wonder why should coming up with explicit construction be so difficult. After all, a proof of existence via the probabilistic method shows not only that an object with the desired property exists but in fact the vast majority of objects have the property."
>
> *Sanjeev Arora and Boaz Barak* [AB09]

1.1 From Circuit Lower Bounds to Range Avoidance

Proving unconditional lower bounds for non-uniform circuits is one of the grand challenges in theoretical computer science, with the holy grail of proving $\mathsf{NP} \not\subseteq \mathsf{P}_{/\mathrm{poly}}$. Unfortunately, progress in unconditional circuit lower bounds has been slow, and the best lower bound for any explicit function against general circuits is only slightly above $3n$ [FGHK16, LY22]. A long-standing, and somewhat embarrassing, open problem is to find any language in $\mathsf{EXP}^{\mathsf{NP}}$ (exponential time with an NP oracle) that cannot be computed by polynomial-size circuits. It seems unlikely that $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{P}_{/\mathrm{poly}}$, but we appear to be very far from ruling out this possibility.

To add more embarrassment, it has been known since 1949 [Sha49] that *most* Boolean functions over n inputs require circuits of size $\Omega(2^n/n)$. 70 years later, we still struggle to spell out even a single such function from a plethora of them! It turns out that circuit lower bounds are not alone, and the difficulty of "finding hay in a haystack" ([AB09, Chapter 21]) is a general phenomenon in theoretical computer science. For example, most graphs are Ramsey graphs [Erd59] and most matrices are rigid matrices [Val77], but it remains major open problems to explicitly construct Ramsey graphs and rigid matrices with good parameters [CZ19, Li23, AC19, Ram20, BHPT20].

Our lack of progress in such explicit construction problems suggests the necessity of a systematic study of their difficulty. As a first step towards building a complexity theory for explicit construction problems, Korten [Kor21] studied the complexity class APEPP defined in [KKMP21], and argued that this is the complexity class that corresponds to explicit construction problems. APEPP is the class of total search problems that are polynomial-time reducible to the following problem:

Problem 1.1 (Range Avoidance Problem, denoted as 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 existence of such y follows from the *dual weak pigeonhole principle*: if we throw 2^n pigeons into 2^ℓ holes, where $\ell \geq n+1$, then there is an empty hole. Thus Avoid is a *total* search problem. Moreover, a random string $y \in \{0,1\}^\ell$ is a valid solution w.p. $1 - 2^{n-\ell} \geq 1/2$, thus there is a trivial randomised algorithm for Avoid. Hence, the focus is to design *deterministic* algorithms for Avoid.

The following is a good example of how Avoid captures the complexity of explicit constructions:

Example 1.2 ([Kor21, Section 3.1])**.** Proving circuit lower bounds can be rephrased as solving the following total search problem, denoted as Hard: On input 1^N where $N = 2^n$, output the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by circuits of size s (say $s = 2^{n/2}$).

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. (The circuit TT is sometimes called the *truth table* generator, hence the name TT.) If we could solve Avoid on the particular instance TT, we would find a truth table $tt \in \{0,1\}^{2^n}$ without size-s circuits, thereby proving a circuit lower bound. It follows that Hard polynomial-time reduces to Avoid, and thus Hard $\in \mathsf{APEPP}$.

More precisely, solving Avoid for TT in polynomial time is equivalent to proving a circuit lower bound for E, and solving Avoid for TT in $\mathsf{FP}^{\mathsf{NP}}$ is equivalent to proving a circuit lower bound for E^{NP}.

1.1.1 Range Avoidance for Restricted Circuit Classes

In a recent paper, Ren, Santhanam, and Wang [RSW22] suggested studying the range avoidance problem for restricted circuit classes. Let $\mathscr{C}$ be a circuit class and $\ell := \ell(n) > n$ be a stretch function. Consider the following problem:

Problem 1.3 ($\mathscr{C}$-Avoid)**.** Given the description of a circuit $C : \{0,1\}^n \to \{0,1\}^{\ell(n)}$, where each output bit of C is a $\mathscr{C}$ circuit, output any string $y \in \{0,1\}^{\ell(n)}$ that is not in the range of C. That is, for every $x \in \{0,1\}^n$, $C(x) \neq y$.

There are lots of reasons for studying the problem $\mathscr{C}$-Avoid, but we only mention one of them here. Many interesting explicit construction problems reduce to $\mathscr{C}$-Avoid for restricted circuit classes $\mathscr{C}$ and (sometimes) large stretch functions ℓ. For example:

- For any "nice" circuit class $\mathscr{C}$, the problem of proving circuit lower bounds against $\mathscr{C}$ can be reduced to $\mathscr{C}$-Avoid via the truth table generator in Example 1.2, where the input of the truth table generator is replaced by a $\mathscr{C}$ circuit (instead of a general circuit).
- Guruswami, Lyu, and Wang [GLW22] showed that the problem of finding rigid matrices and optimal binary linear codes can be reduced to NC^1-Avoid. By a further result in [RSW22], these problems also reduce to NC^0-Avoid. A recent work [GGNS23] showed that the problem of finding rigid matrices can even be reduced to NC^0_3-Avoid.

In general, for any explicit construction problem Π, we can identify a circuit class $\mathscr{C}$ that is as "simple" as possible, as well as a stretch function $\ell(n)$ that is as large as possible, such that Π reduces to $\mathscr{C}$-Avoid with stretch $\ell(n)$. The hope is that by making progress on the range avoidance problem for restricted circuits and by optimising the reduction (i.e., optimising $\mathscr{C}$ and $\ell(n)$), we could solve many explicit construction problems systematically.

An "Algorithmic Method" for range avoidance. Inspired by the Algorithmic Method for proving circuit lower bounds (e.g. [Wil13, Wil14a, MW20, CW19b, Che19, CLW20]), [RSW22] proposed a framework that uses data structures to solve $\mathscr{C}$-Avoid in $\mathsf{FP}^{\mathsf{NP}}$. Consider the following data structure problem.

Problem 1.4 (Hamming Weight Estimation)**.** Let $\mathscr{C}$ be a circuit class and $\ell := \ell(n)$ be a stretch function. The data structure problem has two phases:

(Preprocessing) Given the description of a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where each output bit of C is a $\mathscr{C}$ circuit, we need to preprocess the circuit in P^{NP} (i.e., in polynomial time with an NP oracle) and output a data structure $\mathsf{DS} \in \{0,1\}^{\mathrm{poly}(\ell)}$.

(Query) Given an input x and oracle access (i.e., random access) to DS, we need to estimate the Hamming weight of $C(x)$ in "non-trivial" time, i.e., deterministic $\ell/\log^{\omega(1)} \ell$ time.

It was shown in [RSW22] that for "typical" circuit classes $\mathscr{C}$, a non-trivial data structure for the Hamming Weight Estimation problem for $\mathscr{C}$ implies an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid.

One drawback of [RSW22] is that their framework does not imply new unconditional algorithms for range avoidance.[1] For comparison, the original Algorithmic Method has made significant progress on proving *unconditional* circuit lower bounds that we do not know how to prove otherwise. One motivation for the current paper is to address this drawback by designing new and unconditional

[1] Actually, [RSW22] provided an unconditional range avoidance algorithm for de Morgan formulas with non-trivial parameters. Subsequently, [GLW22] improved this result by using simpler techniques and achieving better parameters; in particular, the algorithm in [GLW22] does not require the Algorithmic Method.

range avoidance algorithms via the Algorithmic Method. In particular, can we solve ACC^0-Avoid with parameters that match the circuit lower bounds in [CLW20]?

1.1.2 The Remote Point Problem

The Algorithmic Method is extremely good at proving average-case circuit lower bounds [CR22, CLW20, CL21]. Therefore, it is natural to wonder if there is an "average-case analogue" of [RSW22].

For two strings $x, y \in \{0,1\}^n$, their *relative Hamming distance* is defined as the fraction of indices where x and y differ, formally $\delta(x,y) := \frac{1}{n}|\{i \in [n] : x_i \neq y_i\}|$. The "average-case analogue" of the range avoidance problem is the following problem:

Problem 1.5 (Remote Point Problem, denoted as $\mathscr{C}$-Remote-Point)**.** Given the description of a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ and a parameter $\delta > 0$, where each output bit of C is a $\mathscr{C}$ circuit, output any string $y \in \{0,1\}^\ell$ that is δ-far from the range of C. That is, for every $x \in \{0,1\}^n$, $\delta(C(x), y) \geq \delta$.

By Chernoff bound, if $\delta < 1/2 - c\sqrt{n/\ell}$ for some absolute constant $c > 0$, then a random length-ℓ string is a valid solution for Remote-Point w.h.p. Therefore, the challenge is to find deterministic algorithms for Remote-Point.

It is not hard to see that $\mathscr{C}$-Remote-Point for the truth table generator TT corresponds to average-case circuit lower bounds. In particular, the regime where δ is a small constant corresponds to proving "weak" average-case lower bounds (e.g. [COS18, Che19]), and the regime where δ is close to $1/2$ (say, $\delta = 1/2 - 1/n$) corresponds to proving "strong" average-case lower bounds (e.g. [CR22, CLW20]).[2]

The remote point problem was discussed in [KKMP21]. Indeed, an important special case of the problem has been studied by Alon, Panigrahy, and Yekhanin [APY09], namely the case that C is a linear transformation over GF(2). In other words, we are given a linear code $C : \{0,1\}^n \to \{0,1\}^\ell$ and we want to find a string far from every codeword. They introduced this problem as an intermediate step towards constructing rigid matrices. In this paper, we call this special case XOR-Remote-Point.

It is already quite hard to solve this special case deterministically. Alon, Panigrahy, and Yekhanin [APY09] designed a polynomial-time algorithm for XOR-Remote-Point when $\ell > 2n$ and $\delta = O(\log n/n)$. For slightly larger δ, say $\delta = 0.1$, no deterministic algorithm is known even with an NP oracle. Arvind and Srinivasan [AS10] showed that for certain parameters, a polynomial-time algorithm for XOR-Remote-Point implies a polynomial-time algorithm for AC^0-Partial-Hard (defined later in Section 1.1.3).

1.1.3 Hard Partial Truth Tables

Besides Avoid and Remote-Point, we also consider the following problem that generalises the task of proving circuit lower bounds (in a different way from Avoid and Remote-Point):

Problem 1.6 (Hard Partial Truth Tables against $\mathscr{C}$, denoted as $\mathscr{C}$-Partial-Hard)**.** Given a list of input strings $z_1, z_2, \ldots, z_\ell \in \{0,1\}^n$ and a parameter s, find a list of output bits $b_1, b_2, \ldots, b_\ell \in \{0,1\}$ such that the partial function defined by $\{(z_i, b_i)\}_{i \in [\ell]}$ cannot be computed by $\mathscr{C}$ circuits of size s. In other words, for every size-s $\mathscr{C}$ circuit C, there exists an index $i \in [\ell]$ such that $C(z_i) \neq b_i$.

[2] Typically, a strong average-case lower bound states that certain problems cannot be $(1/2 + 1/s)$-approximated by size-s circuits. Suppose $\mathsf{TT} : \{0,1\}^n \to \{0,1\}^\ell$ is the truth table generator, then n is roughly the size of the circuit (i.e., $n \approx s$). In this regard, strong average-case circuit lower bounds correspond to Remote-Point where $\delta = 1/2 - 1/n$.

It is easy to see that $\mathscr{C}$-Partial-Hard generalises the problem of proving circuit lower bounds against $\mathscr{C}$. Indeed, if we take $\ell := 2^n$ and $z_1, z_2, \ldots, z_\ell$ be an enumeration of length-n strings, then $\mathscr{C}$-Partial-Hard becomes exactly the problem of proving circuit lower bounds against $\mathscr{C}$. It is also easy to see that when $\ell > O(s \log s)$, this problem is in APEPP: given the input $(z_1, z_2, \ldots, z_\ell)$, we can construct a circuit $\mathsf{TT}' : \{0,1\}^{O(s \log s)} \to \{0,1\}^\ell$ which takes the description of a $\mathscr{C}$ circuit C as input, and outputs the concatenation of $C(z_1), C(z_2), \ldots, C(z_\ell)$. Finding a non-output of TT' is equivalent to finding a solution of $\mathscr{C}$-Partial-Hard, thus this problem reduces to Avoid.

This problem was introduced by Arvind and Srinivasan [AS10] under the name "circuit lower bounds with help functions." Let $h_1, h_2, \ldots, h_n : \{0,1\}^m \to \{0,1\}$ denote a sequence of *help functions*, $\mathscr{C}$ be a circuit class, and $s \in \mathbb{N}$ be a size parameter. The goal is to construct the truth table of a function $f : \{0,1\}^m \to \{0,1\}$ that is hard to compute for size-s $\mathscr{C}$ circuits, even when the circuit has access to these help functions. Formally, for any size-s circuit $C : \{0,1\}^n \to \{0,1\}$, there exists an input $x \in \{0,1\}^m$ such that

$$C(h_1(x), h_2(x), \ldots, h_n(x)) \neq f(x).$$

This problem is equivalent to Partial-Hard with $\ell = 2^m$ inputs of length n, namely for every $x \in \{0,1\}^m$, there is an input $h_1(x) \circ h_2(x) \circ \cdots \circ h_n(x) \in \{0,1\}^n$ in the Partial-Hard instance.

This problem appears to be very hard. Neither [AS10] nor we are aware of an efficient deterministic solution for $\mathscr{C} = \mathsf{AC}^0$ with (say) $\ell, s \in$ quasi-poly(n). That is, although exponential-size lower bounds against AC^0 are known [Ajt83, FSS84, Yao85, Hås89], we do not have any idea about how to prove such a lower bound for partial functions. Even when $\mathscr{C}$ is the class of *polynomial-size DNF*, to the best of our knowledge, there is no known deterministic algorithm for $\mathscr{C}$-Partial-Hard.

Besides being a natural problem itself, $\mathscr{C}$-Partial-Hard also arises when we study the closure of non-uniform complexity classes (under reductions). Recall that AC^0 denotes the class of languages computable by a *non-uniform* family of polynomial-size constant-depth circuits; in particular, AC^0 contains undecidable languages such as unary versions of the halting problem. A language L Turing-reduces to some language in AC^0 if and only if $L \in \mathsf{P}_{/\mathrm{poly}}$ [Pip79], thus proving $\mathsf{EXP} \not\leq^p_T \mathsf{AC}^0$ is likely beyond current techniques. But what about *mapping reducibility*? Can we show that $\mathsf{EXP} \not\leq^p_m \mathsf{AC}^0$? It turns out that a deterministic algorithm for AC^0-Partial-Hard implies that $\mathsf{EXP} \not\leq^p_m \mathsf{AC}^0$ [AS10, Theorem 5]. Of course, there is nothing special with AC^0, and it can be replaced by other non-uniform classes. Therefore, $\mathscr{C}$-Partial-Hard sheds light on ruling out many-one reducibility of EXP (and other complexity classes) to non-uniform classes.

We can also define the average-case version of $\mathscr{C}$-Partial-Hard, which is equivalent to proving average-case lower bounds with help functions.

Problem 1.7 (Average-Case Hard Partial Truth Tables against $\mathscr{C}$, denoted as $\mathscr{C}$-Partial-AvgHard)**.** Given a list of input strings $z_1, z_2, \ldots, z_\ell \in \{0,1\}^n$ and parameters s, δ, find a list of output bits $b_1, b_2, \ldots, b_\ell \in \{0,1\}$ such that the partial function defined by $\{(z_i, b_i)\}_{i \in [\ell]}$ is δ-far from being computable by $\mathscr{C}$ circuits of size s. In other words, for every size-s $\mathscr{C}$ circuit C, there are at least $\delta\ell$ indices $i \in [\ell]$ such that $C(z_i) \neq b_i$.

1.2 Our Results

In this sub-section, we describe our results in detail.

1.2.1 Explicit Constructions from Satisfying-Pairs Algorithms

We start with the following observation: In the framework of solving Avoid via the Algorithmic Method [RSW22], the data structure for Problem 1.4 does not need to be *online*. Instead, it suffices

to design a data structure that preprocesses a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, receives a *batch* of inputs $x_1, x_2, \ldots, x_M$, and estimates the Hamming weight of each $C(x_i)$ in *non-trivial total time*, i.e., $\ell M/\log^{\omega(1)}(\ell M)$ time. Moreover, we observe that it is not even necessary to estimate the individual Hamming weights $C(x_i)$; it suffices to estimate the *average* Hamming weight of $C(x_i)$ for $i \in [M]$. Indeed, we arrive at the following problem called *Satisfying Pairs*.[3]

Problem 1.8 ($\mathscr{C}$-Satisfying-Pairs)**.** Let N, M, s, n be parameters. Given (single-output) $\mathscr{C}$ circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s and input strings $x_1, x_2, \ldots, x_M \in \{0,1\}^n$, compute or estimate

$$\Pr_{i \leftarrow [M], j \leftarrow [N]}[C_j(x_i) = 1]. \tag{1}$$

We define the decisional and counting versions of the satisfying pairs problem as follows.

- Gap_δ-$\mathscr{C}$-Satisfying-Pairs is the problem of distinguishing between (1) $= 1$ and (1) $< 1 - \delta$;
- $\mathsf{Approx}_\varepsilon$-$\mathscr{C}$-Satisfying-Pairs is the problem of estimating (1) within additive error ε;
- $\mathscr{C}$-Satisfying-Pairs is the problem of deciding whether (1) > 0;
- $\#\mathscr{C}$-Satisfying-Pairs is the problem of exactly computing (1).

We consider the regime where the input length n and the circuit size s are much smaller than N and M. In such case, a deterministic algorithm for $\mathscr{C}$-Satisfying-Pairs is said to be *non-trivial* if it runs in time $NM/\log^{\omega(1)}(NM)$.[4]

Remark 1.9. The circuit-analysis problems that arise in the Algorithmic Method[5] are special cases of Satisfying Pairs problems. For instance, we can solve #SAT of the circuit C by solving #Satisfying-Pairs with $N = 2^{n/2}$ and $M = 2^{n/2}$, where the inputs $(x_1, x_2, \ldots, x_M)$ consists of all strings of length $n/2$, and the circuits are $\{C_y : y \in \{0,1\}^{n/2}\}$, where $C_y(x) := C(x \circ y)$. Similarly, $\mathscr{C}$-Satisfying-Pairs corresponds to $\mathscr{C}$-SAT, Gap-$\mathscr{C}$-Satisfying-Pairs corresponds to $\mathscr{C}$-GapUNSAT, and Approx-$\mathscr{C}$-Satisfying-Pairs corresponds to $\mathscr{C}$-CAPP.

Range Avoidance from Satisfying-Pairs. By plugging the observation above in [RSW22], we show that non-trivial algorithms for Satisfying-Pairs imply $\mathsf{FP}^{\mathsf{NP}}$ algorithms for Avoid.

Theorem 1.10 (Theorem 3.2, Informal)**.** *Let $\mathscr{C}$ be a typical circuit class and $\mathscr{C}' := \mathsf{OR}_2 \circ \mathscr{C}$.*[6] *Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for every constant $\varepsilon > 0$, then $\mathscr{C}$-Avoid with certain parameters can be solved in $\mathsf{FP}^{\mathsf{NP}}$.*

This informal version of Theorem 3.2 hides the trade-off between the parameters of $\mathscr{C}$-Avoid and $\mathscr{C}'$-Satisfying-Pairs. In general, to solve $\mathscr{C}$-Avoid with smaller stretch ℓ (with respect to the input length n), we need to have non-trivial algorithms for $\mathscr{C}'$-Satisfying-Pairs where the

[3] We remark that our definition of $\mathscr{C}$-Satisfying-Pairs is different from the fine-grained complexity literature (e.g., [AHWW16, CW19a]). The input of the $\mathscr{C}$-Satisfying-Pairs problem defined in [AHWW16, CW19a] consists of a circuit $C(-,-)$ and two sets of input strings $\{a_i\}$ and $\{b_j\}$, and one wants to compute or approximate the number of pairs (i,j) such that $C(a_i, b_j) = 1$; in our $\mathscr{C}$-Satisfying-Pairs problem, we receive as input a list of circuits $\{C_i\}$ and a list of inputs $\{x_j\}$, and we want to compute or approximate the number of pairs (i,j) such that $C_i(x_j) = 1$. The new definition fits our purpose better. We also remark that for circuit classes that can "evaluate themselves" (such as $\mathsf{AC}^0, \mathsf{ACC}^0$, and TC^0), these two definitions are computationally equivalent.

[4] Analogous to the preprocessing phase in Problem 1.4, one could also add a P^{NP}-preprocessing phase that sees the circuits but not the inputs. Algorithms with such preprocessing phase would still imply our results, but the Satisfying-Pairs algorithms in this paper do not need this preprocessing phase.

[5] The definitions of circuit-analysis problems such as SAT or CAPP can be found in Lijie Chen's PhD thesis [Che22].

[6] Here, $\mathsf{OR}_d \circ \mathscr{C}$ refers to the composition of a single fan-in-d OR gate being the output gate of the circuit and (at most) d $\mathscr{C}$ circuits feeding the top OR gate.

number of inputs N and the number of circuits M are smaller with respect to the circuit size s and the input length n. We highlight two typical choices of parameters of Theorem 3.2 as follows.

Corollary 1.11. *There is a constant $\varepsilon > 0$ such that the following holds. Let $\mathscr{C}$ be a typical circuit class, $\mathscr{C}' := \mathsf{OR}_2 \circ \mathscr{C}$, and $s = s(n)$ be a non-decreasing size parameter.*

- *Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for $N = n^{1+\Omega(1)}$ $\mathscr{C}'$-circuits of size $2s(n)$ and $M = n^{1+\Omega(1)}$ inputs of length n. Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid with stretch ℓ and circuit size s,[7] for some $\ell = n^{1+\Omega(1)}$.*
- *Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for $N = \text{quasi-poly}(n)$ $\mathscr{C}'$-circuits of size $2s(n)$ and $M = \text{quasi-poly}(n)$ inputs of length n. Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid with stretch ℓ and circuit size s, for some $\ell = \text{quasi-poly}(n)$.*

Remote Point from Satisfying-Pairs. With the help of a smooth and rectangular PCPP (see Section 6) and a linear-sum list-decodable code from [CLW20] (also see Appendix A.5), we show that non-trivial algorithms for Satisfying-Pairs imply $\mathsf{FP}^{\mathsf{NP}}$ algorithms for Remote-Point.

Theorem 1.12 (Theorem 3.5, Informal)**.** *Let $\mathscr{C}$ be a typical circuit class and $\mathscr{C}' := \mathsf{AND}_{O(1)} \circ \mathscr{C}$. Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for every constant $\varepsilon > 0$, then $\mathscr{C}$-Remote-Point with certain parameters can be solved in $\mathsf{FP}^{\mathsf{NP}}$.*

In particular: suppose for every constant $\varepsilon > 0$, there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for $N = \text{quasi-poly}(n)$ $\mathscr{C}'$-circuits of size $O(s)$ and $M = \text{quasi-poly}(n)$ inputs of length n; then for some stretch function $\ell = \text{quasi-poly}(n)$, there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Remote-Point that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ where each output bit of C is a $\mathscr{C}$-circuit of size s, and outputs a y that is 0.49-far from $\mathsf{Range}(C)$.

Our framework provides Remote-Point algorithms for the regime corresponding to "strong average-case lower bounds", i.e., the distance between the output y and $\mathsf{Range}(C)$ is close to $1/2$. In fact, the distance can be as large as $1/2 - 1/\text{poly}(n)$ given an Approx-$\mathscr{C}$-Satisfying-Pairs algorithm with a small enough error. (see Theorem 3.5 for details).

Note that the stretch for $\mathscr{C}$-Remote-Point that we can solve in $\mathsf{FP}^{\mathsf{NP}}$ depends on both the parameters of the satisfying pairs algorithms and the rate of the linear-sum list-decodable code. Since the code from [CLW20] has a quasi-polynomial rate, our framework cannot solve Remote-Point with stretch smaller than quasi-polynomial. It is an interesting open problem to improve the stretch of Remote-Point that can be solved by our framework, possibly by designing new linear-sum decodable codes with a better rate; see, e.g., [CL21].

Hard Partial Truth Table from Satisfying-Pairs. Similar to the frameworks for Avoid and Remote-Point, we can solve Partial-Hard and Partial-AvgHard via non-trivial algorithms for Satisfying-Pairs.

Theorem 1.13 (Theorems 4.2 and 4.3, Informal)**.** *Let $\mathscr{C}$ be a typical circuit class.*

- *Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for every $\varepsilon > 0$ and $\mathscr{C}' := \mathsf{OR}_2 \circ \mathscr{C}$, then $\mathscr{C}$-Partial-Hard with certain parameters can be solved in $\mathsf{FP}^{\mathsf{NP}}$.*
- *Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}''$-Satisfying-Pairs for every $\varepsilon > 0$ and $\mathscr{C}'' := \mathsf{AND}_{O(1)} \circ \mathscr{C}$, then $\mathscr{C}$-Partial-AvgHard with certain parameters can be solved in $\mathsf{FP}^{\mathsf{NP}}$.*

[7] Note that the circuit size parameter of $\mathscr{C}$-Avoid refers to the maximum circuit size of each output bit of $C : \{0,1\}^n \to \{0,1\}^\ell$, instead of the total circuit size of C.

These results are proved using essentially the same approach as the framework for Avoid and Remote-Point; subsequently, the trade-off between parameters for Satisfying-Pairs and Partial-Hard (resp. Partial-AvgHard) is similar to that for Avoid (resp Remote-Point). We omit the details and refer the readers to Theorems 4.2 and 4.3.

Remark 1.14. It is not surprising to have a unified framework for Avoid and Partial-Hard (and their average-case analogues Remote-Point and Partial-AvgHard), since Avoid and Partial-Hard can be considered as the *dual* problem of each other. Let $\mathsf{Eval} : \{0,1\}^{O(s \log s)} \times \{0,1\}^n \to \{0,1\}$ be the circuit-evaluation function that takes a circuit C of size s and an input of length n, and outputs $C(x)$. We can interpret Avoid and Partial-Hard as follows:

- **(Avoid).** Given size-s circuits $C_1, C_2, \ldots, C_\ell$, find $y_1, y_2, \ldots, y_\ell \in \{0,1\}$ such that for every $x \in \{0,1\}^n$, there is an $i \in [\ell]$ such that $\mathsf{Eval}(C_i, x) \neq y_i$.
- **(Partial-Hard).** Given inputs $x_1, x_2, \ldots, x_\ell \in \{0,1\}^n$, find $y_1, y_2, \ldots, y_\ell \in \{0,1\}$ such that for every size-s circuit C, there is an $i \in [\ell]$ such that $\mathsf{Eval}(C, x_i) \neq y_i$.

Clearly, Avoid and Partial-Hard are essentially the same problem on the table $\mathsf{Eval}(\cdot,\cdot)$ with the rows and columns being exchanged.

1.2.2 Unconditional Results for Explicit Constructions

The seemingly marginal improvement of using Satisfying-Pairs instead of its online version Hamming Weight Estimation (see Problem 1.4) plays an important role in the design of unconditional $\mathsf{FP}^{\mathsf{NP}}$ algorithms for ACC^0-Remote-Point and ACC^0-Partial-Hard, because we can indeed design non-trivial algorithms for ACC^0-Satisfying-Pairs.

XOR-Remote-Point from XOR-Satisfying-Pairs. We start from a simpler case where the circuit class $\mathscr{C} = \mathsf{XOR}$, i.e., the circuit is an XOR of some of its input bits. Since an XOR circuit C can be represented by a vector $\vec{v} \in \{0,1\}^n$ such that $C(x) = \langle v, x \rangle \bmod 2$, #XOR-Satisfying-Pairs is nothing but the counting version of the *Orthogonal Vector* problem over $\mathbb{F}_2$, which admits a non-trivial algorithm [CW21, AC19]. By combining this with Theorem 1.10, we obtain an unconditional $\mathsf{FP}^{\mathsf{NP}}$ algorithm for XOR-Remote-Point.[8]

Theorem 1.15 (XOR-Remote-Point $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \geq 1$ such that the following holds. Let $\varepsilon := \varepsilon(n) \geq 2n^{-c_u}$ be the error parameter and $\ell := \ell(n) \geq 2^{\log^{c_u+5} n}$ be the stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where each output bit of C is computed by an XOR gate, and outputs a string y that is $(1/2 - \varepsilon)$-far from $\mathsf{Range}(C)$.*

A non-trivial algorithm for ACC^0-Satisfying-Pairs. By slightly adapting the technique introduced by Williams [Wil18c] to design non-trivial #SAT algorithms for ACC^0 circuits with an earlier quasi-polynomial size simulation of $\mathsf{SYM} \circ \mathsf{ACC}^0$ circuits by $\mathsf{SYM} \circ \mathsf{AND}$ circuits [BT94, AG91], we can obtain a non-trivial algorithm for #ACC^0-Satisfying-Pairs, formally stated as follows.

Theorem 1.16. *For every constants m, ℓ, c, there is a constant $\varepsilon \in (0,1)$ such that the following holds. Let $n := 2^{\log^\varepsilon N}$ and $s := 2^{\log^c n}$. There is a deterministic algorithm running in $\tilde{O}((N/n)^2)$ time that given N strings $x_1, x_2, \ldots, x_N \in \{0,1\}^n$ and N $\mathsf{AC}^0_\ell[m]$ circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s, outputs the number of pairs $(i,j) \in [N] \times [N]$ such that $C_i(x_j) = 1$.*

[8] The reduction from Remote-Point to Satisfying-Pairs has a small overhead on the circuit class (i.e. the upper $\mathsf{AND}_{O(1)}$ in Theorem 1.12). By a standard trick using Fourier analysis (see Theorem 2.17; also see [CW19b]), we can change the upper circuit class to be $\mathsf{XOR}_{O(1)}$ so that we only need to design Satisfying-Pairs algorithms for $\mathsf{XOR}_d \circ \mathsf{XOR} = \mathsf{XOR}$.

Explicit constructions for ACC^0. The $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Remote-Point and ACC^0-Partial-AvgHard follows from this algorithm together with Theorem 1.12 and Theorem 1.13.

Theorem 1.17 (ACC^0-Remote-Point $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \geq 1$ such that for every constant $d, m \geq 1$, there is a constant $c_{\mathsf{str}} := c_{\mathsf{str}}(d, m) \geq 1$, such that the following holds.*

Let $n < s(n) \leq 2^{n^{o(1)}}$ be a size parameter, $\varepsilon := \varepsilon(n) \geq 2n^{-c_u}$ be an error parameter and $\ell := \ell(n) \geq 2^{\log^{c_{\mathsf{str}}} s}$ be a stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where each output bit of C is computed by an $\mathsf{AC}^0_d[m]$ circuit of size s, and outputs a string y that is $(1/2 - \varepsilon)$-far from $\mathsf{Range}(C)$.

Theorem 1.18 (ACC^0-Partial-AvgHard $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \geq 1$ such that for every constants $d, m \geq 1$, there is a constant $c_{\mathsf{str}} := c_{\mathsf{str}}(d, m) \geq 1$, such that the following holds.*

Let $n < s(n) \leq 2^{n^{o(1)}}$ be a size parameter, $\varepsilon := \varepsilon(n) \geq 2n^{-c_u}$ be an error parameter and $\ell := \ell(n) \geq 2^{\log^{c_{\mathsf{str}}} s}$ be a stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that given inputs $x_1, \ldots, x_\ell \in \{0,1\}^n$, it outputs a string $y \in \{0,1\}^\ell$ such that for any $s(n)$-size $\mathsf{AC}^0_d[m]$ circuit C, y is $(1/2-\varepsilon)$-far from $C(x_1) \circ \cdots \circ C(x_\ell)$.

It is worth noting that the ACC^0-Remote-Point algorithm here recovers the best known almost-everywhere average-case circuit lower bounds against ACC^0 [CLW20]. This is done by considering the special case where the input circuit is the truth table generator $\mathsf{TT} : \{0,1\}^{O(s \log s)} \to \{0,1\}^{2^n}$ that prints the truth table of a given ACC^0 circuit (see Section 5.3).

Corollary 1.19. *For every constants $d, m \geq 1$, there is an $\varepsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L_n cannot be $(1/2 + 2^{-n^\varepsilon})$-approximated by $\mathsf{AC}^0_d[m]$ circuits of size 2^{n^ε}, for all sufficiently large n.*

Lower bounds on the many-one closure of ACC^0. Following the observation of Arvind and Srinivasan [AS10], the $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Partial-AvgHard can be used to prove *unconditionally* that E^{NP} cannot be mapping reduced to languages decidable by small-size *non-uniform* families of ACC^0 circuits.[9] To the best of our knowledge, this is the first unconditional result ruling out the mapping reducibility from uniform classes to non-trivial non-uniform classes.

Corollary 1.20. *Let $d, m \in \mathbb{N}$ be constants, $\mathsf{AC}^0_d[m]$ denote the class of languages computable by a* non-uniform *family of polynomial-size $\mathsf{AC}^0_d[m]$ circuits. Then, there is a language $L^{\mathsf{hard}} \in \mathsf{E}^{\mathsf{NP}}$ that does not have polynomial-time mapping reductions to any language in $\mathsf{AC}^0_d[m]$.*

1.2.3 A Smooth and Rectangular PCPs of Proximity

One of the main technical ingredients in our framework for the average-case explicit construction problems (i.e. Remote-Point and Partial-AvgHard) is a PCP of Proximity (PCPP) that is *short*, *smooth*, and (almost) *rectangular*.

A PCPP verifier V for a language L provides a super-efficient probabilistic proof system for checking whether $x \in L$ or x is far from being in L. Given an input x and a proof π, the verifier with access to some random bits only probes constantly many bits of x and π. If $x \in L$, then it accepts with an appropriate proof π; if the relative Hamming distance between x and any $x' \in L$ is at least δ, then it rejects with constant probability regardless of the proof π. (The distance δ is called the *proximity parameter* of the PCPP.) In addition, our PCPP verifier is equipped with the following properties:

[9] In fact, it suffices to have an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Partial-Hard (which is a trivial consequence of an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Partial-AvgHard) for this application.

- **(Shortness).** For any language $L \in \mathsf{NTIME}[T(n)]$, where $n \leq T(n) \leq 2^{\mathrm{poly}(n)}$, the PCPP proof for L has length $T(n) \cdot \mathrm{polylog}(T(n))$.
- **(Rectangularity).** The input and the proof are treated as matrices. Moreover, the queries of the verifier to the input and proof matrices can be done *rectangularly*, in the sense that there are a row verifier V_{row} and a column verifier V_{col} with (almost) independent random seeds that generate the row and column indices of the queries, respectively.
- **(Smoothness).** The queries of the verifier to the proof matrix are uniformly random. As a consequence, it means that the PCPP proof can tolerate errors in a correct proof.

We refer the readers to Section 2.5 for formal definitions of these properties.

Before our work, Bhangale, Harsha, Paradise, and Tal [BHPT20] constructed a short, smooth, and rectangular PCP (instead of PCPP) built upon [BGH+06] with an application of constructing *rigid matrices* (also see [Val77, AC19]). Ren, Santhanam, and Wang [RSW22] constructed a short and rectangular PCPP based on [BGH+06,BHPT20] for the Algorithmic Method for Avoid. It turns out that to generalise [RSW22] to the "average-case" explicit construction problems Remote-Point and Partial-AvgHard, we need both *smoothness* (as in [BHPT20]) and PCPs of *proximity* (as in [RSW22]). A technical contribution of this work is to combine [BHPT20] and [RSW22] to obtain a *smooth PCPP*.

Theorem 1.21 (Theorem 2.14, Informal)**.** *Let $T(n)$ be a good function. For every language $L \in \mathsf{NTIME}[T(n)]$, there is a short, smooth, and (almost) rectangular PCP of proximity verifier V for L, with perfect completeness, constant soundness error, and constant query complexity.*

Following standard techniques in the algorithmic approach to lower bounds (see, e.g., [CW19b]), we also construct a short and rectangular (non-smooth) PCPP that makes at most two queries to the input and the proof matrices (see Theorem 2.13) to minimise the overhead on the circuit class when we reduce Avoid and Partial-Hard to Satisfying-Pairs (i.e. the upper OR_2 in Theorem 1.10 and Theorem 1.13).

1.3 Technical Overview

As mentioned in Section 1.2.1, the range avoidance algorithm follows from slightly modifying the framework in [RSW22] and using an algorithm for Satisfying-Pairs. In what follows, we briefly illustrate our techniques for the remote point problem and for constructing hard partial truth tables. The high-level idea is to reduce these problems to Avoid and invoke our framework for Avoid to solve them.

Remote Point Problem. Our start point is the following reduction from Remote-Point to Avoid. Suppose that $C : \{0,1\}^n \to \{0,1\}^\ell$ is the input circuit. Let $\mathsf{Enc} : \{0,1\}^{\ell'} \to \{0,1\}^\ell$ be the encoding procedure of an error correcting code, and $\mathsf{Dec} : \{0,1\}^\ell \to \{0,1\}^{\ell'}$ be the corresponding decoding procedure, where Dec can correct a δ fraction of errors. Define the circuit $C'(x) := \mathsf{Dec}(C(x))$, and let z be any string not in the range of C', then $\mathsf{Enc}(z)$ is $(1-\delta)$-far from $\mathsf{Range}(C)$. To see this, assume for contradiction that $\mathsf{Enc}(z)$ is $(1-\delta)$-close to some $C(x)$, then $\mathsf{Dec}(C(x))$ should return exactly z, contradicting that z is a non-output of C'.

Suppose that the function Dec can be implemented in the circuit class $\mathscr{C}_{\mathsf{Dec}}$, then this is a reduction from $\mathscr{C}$-Remote-Point to $(\mathscr{C}_{\mathsf{Dec}} \circ \mathscr{C})$-Avoid. Therefore, we would like the complexity of $\mathscr{C}_{\mathsf{Dec}}$ to be as small as possible. There are decoders that tolerate a small constant fraction of errors in AC^0 [GGH+07], so it might be possible to implement $\mathscr{C}_{\mathsf{Dec}}$ in AC^0. However, when δ is very close to $1/2$ (say $\delta = 1/2 - \varepsilon$), we enter the list-decoding regime where $\mathscr{C}_{\mathsf{Dec}}$ seems to need the

power of majority [GR08]. Can we solve $\mathscr{C}$-Remote-Point without invoking any circuit-analysis algorithms for $\mathsf{MAJ} \circ \mathscr{C}$?

Fortunately, the required techniques already appeared in previous works on the Algorithmic Method for proving strong average-case circuit lower bounds. In [CLW20], they provided an error-correcting code that corrects a $1/2 - \varepsilon$ fraction of errors, where the decoder $\mathsf{Dec}_{\mathsf{CLW}}$ can be implemented as a *linear sum*, i.e., each output is a linear combination of the input bits.[10] Intuitively, this means that we can reduce $\mathscr{C}$-Remote-Point to $(\mathsf{Sum} \circ \mathscr{C})$-Avoid, where Sum denotes the layer of $\mathsf{Dec}_{\mathsf{CLW}}$. Using the framework for range avoidance established above, it suffices to solve the Satisfying-Pairs problem for $\mathsf{Sum} \circ \mathscr{C}$ circuits.[11] But it is easy to see that Satisfying-Pairs for $\mathsf{Sum} \circ \mathscr{C}$ circuits directly reduces to Satisfying-Pairs for $\mathscr{C}$ circuits! Therefore, the error-correcting code in [CLW20] allows us to use an algorithm for $\mathscr{C}$-Satisfying-Pairs to directly solve $\mathscr{C}$-Remote-Point, with little or no circuit complexity overhead.

The above discussion omitted several important technical details:

- It turns out that $\mathsf{Dec}_{\mathsf{CLW}}$ is only an *approximate* list-decoding algorithm: given a corrupted codeword that is $(1/2 - \varepsilon)$-close to the correct codeword, we can only recover a message that is δ-close to the correct message (instead of perfectly recovering the correct message).

  This drawback is handled by *smooth PCPPs* [Par21], which has the property that any slightly corrupted version of a correct proof is still accepted with good probability. As we need a rectangular PCPP in [RSW22], what we actually need is a *smooth and rectangular PCPP*, which we construct in Section 6. We remark that [CLW20] also encountered this difficulty; they got around it by combining a PCP and a PCPP for Circuit-Eval. It is not clear how to generalise this strategy to our case.

- Another technical complication is that $\mathsf{Dec}_{\mathsf{CLW}}$ outputs *real values* instead of Boolean values. It is only guaranteed that the decoded message is close to the original message *in ℓ_1-norm*. Consequently, after guessing the PCPP proof, we also need to verify that it is "close to Boolean", This difficulty also appears in [CLW20]; however, we need to carefully define what it means by "close to Boolean" in our case.

- Since $\mathsf{Dec}_{\mathsf{CLW}}$ works in the list-decoding regime, it also receives an advice string (specifying the index of the codeword in the list). In the above discussion, we omitted the advice string to highlight the main ideas. It turns out that the dependency of the decoder on the advice string *cannot* be captured by linear sums. Therefore, we need to define an ad hoc "linear sum" circuit class (in Section 2.4) that receives both an input and an advice string and computes a linear combination over the input, where the "linear combination" depends on the advice. It turns out that we need the dependency on the advice to be *local* (see Section 2.4 for details), which is fortunately satisfied by the code in [CLW20].

Another reduction via succinct dictionaries. We mention that there is another reduction from Remote-Point to Avoid which appears in [Kor21, GLW22]. Let $C : \{0,1\}^n \to \{0,1\}^\ell$ be a circuit, $y \in \{0,1\}^\ell$ be a string that is not δ-far from $\mathsf{Range}(C)$. Then we can find a string $x \in \{0,1\}^n$ and a "noise" string $e \in \{0,1\}^m$ of relative Hamming weight at most δ such that $y = C(x) \oplus e$,

[10] [CLW20] stated this result as a non-standard XOR lemma in their Appendix A. We re-prove it in the form of error-correcting codes in Appendix A.5.

[11] We made a simplification here. Actually, we need to solve Satisfying-Pairs for $\mathsf{NC}^0 \circ \mathsf{Sum} \circ \mathscr{C}$ circuits. Using the distributive property, we can push the NC^0 circuits below the Sum layer, thus it suffices to solve Satisfying-Pairs for $\mathsf{Sum} \circ \mathsf{NC}^0 \circ \mathscr{C}$ circuits. In this informal exposition, we may assume that $\mathscr{C}$ is closed under top NC^0 gates, which means that a Satisfying-Pairs algorithm for $\mathsf{Sum} \circ \mathscr{C}$ now suffices.

where $\oplus$ refers to bit-wise XOR. Consider the circuit $C'(x, e) := C(x) \oplus e$. To solve the remote point problem for C, it suffices to solve the range avoidance problem for C'. Using a "succincter" dictionary to represent e [Păt08], [GLW22] managed to show that this reduction also preserves circuit complexity, and in particular reduces NC^1-Remote-Point to NC^1-Avoid.

A drawback of this approach is that it only reduces Remote-Point to range avoidance instances with a small stretch. Indeed, suppose C' is a circuit from n' inputs to ℓ outputs, and $\delta = \Omega(1)$, then

$$n' \geq |\Pi(e)| \geq \log \binom{\ell}{\delta\ell} = \Omega(\ell).$$

In contrast, the algorithmic method in both [RSW22] and this paper could not solve range avoidance instances with such a small stretch ($\ell = c \cdot n$ for some constant c), even with the best possible algorithms for Satisfying-Pairs. Therefore we do not use this approach in this paper.

Hard Partial Truth Table. There is a simple reduction from Partial-Hard to Avoid. Suppose we are given strings $x_1, x_2, \ldots, x_N$. Let TT' be the circuit that receives a size-s circuit C as input, and outputs the concatenation of $C(x_1)$, $C(x_2)$, $\ldots$, $C(x_N)$. If $N > O(s \log s)$ then the circuit TT' is stretching. It is also easy to see that solving the range avoidance of TT' is equivalent to solving the Partial-Hard problem.

In Section 4, we essentially combine this reduction with the frameworks in Section 3. In other words, we could have reduced Partial-Hard to Avoid in a black-box way and derived the main results in Section 4. However, this reduction only reduces $\mathscr{C}$-Partial-Hard to $\mathscr{C}'$-Avoid, where $\mathscr{C}'$ is any circuit class that can solve $\mathscr{C}$-Eval in the following sense: for every fixed input x, there is a $\mathscr{C}'$ circuit C' that takes as input the description of a $\mathscr{C}$ circuit C, and outputs $C(x)$. For most circuit classes of interest (e.g., $\mathscr{C} \in \{\mathsf{AC}^0, \mathsf{ACC}^0, \mathsf{NC}^1, \mathsf{P}_{/\mathrm{poly}}\}$), we could simply let $\mathscr{C}' = \mathscr{C}$; however, this is not necessarily true for more refined circuit classes (such as $\mathscr{C} = \mathsf{ACC} \circ \mathsf{THR}$). We choose to derive the main results in Section 4 from scratch instead of reducing it to Section 3, partly because we also want our framework to hold for these more refined circuit classes.

1.4 Further Related Work

Satisfying-Pairs and the Polynomial Method. The Satisfying-Pairs problems for restricted circuit classes nicely capture a wide range of algorithmic problems that have been extensively studied. For instance, the Orthogonal Vector Problem over $\mathbb{F}_2$ corresponds to XOR-Satisfying-Pairs, and the (decision version of) Nearest Neighbor Problem corresponds to the Satisfying-Pairs of polynomial threshold functions (see, e.g., [Wil14b, ACW16]).

There is a successful line of research on non-trivial algorithms for this kind of problems via the *polynomial method* [Raz87, Smo87] in circuit complexity. Williams [Wil18a] developed an $n^3/2^{(\log n)^{\Omega(1)}}$-time algorithm for the All-Pairs Shortest Path problem using the Razborov-Smolensky polynomial representation of $\mathsf{AC}^0[p]$ circuits [Raz87, Smo87, Smo93] and a fast batch evaluation of polynomials via fast rectangular matrix multiplication [Cop82] (also see Theorem 5.1). Similar techniques were used to design non-trivial algorithms for the Orthogonal Vector Problem over $\mathbb{F}_2$ [CW21, AWY15] and the (approximate) Nearest Neighbor Problems (with respect to Hamming distance, ℓ_1-distance, and ℓ_2-distance) [AW15, ACW16, ACW20]. Chen and Wang [CW19a] (following [AW17]) generalised the polynomial method in algorithm design by showing a connection between Satisfying-Pairs problems and quantum communication protocols, with an application in $\mathsf{Approx}_{\varepsilon}$-$\mathsf{XOR}$-Satisfying-Pairs (which is called Approximate #OV in [CW19a]).

Explicit obstructions. Related to the Partial-Hard problem is the notion of explicit obstructions [Mul11, CJW20]: on input 1^n, one wants to output a list of (x_i, y_i) deterministically, such that $x_i \neq x_j$ for $i \neq j$, and for all n-input circuit C from a certain circuit class $\mathscr{C}$, there is some i such that $C(x_i) \neq y_i$. This notion is weaker than deterministic algorithms for Partial-Hard, as one has the freedom of choosing the inputs $\{x_i\}$. Chen, Jin, and Williams [CJW20] exhibited a "sharp threshold" phenomenon for explicit obstructions against de Morgan formulas: an explicit obstruction for $\mathsf{Formula}[n^{1.99}]$ provably exists, while an explicit obstruction for $\mathsf{Formula}[n^{2.01}]$ would imply very strong circuit lower bounds.

1.5 Organisation

Organisation of the Paper

- In Section 2, we introduce concepts and tools used in this paper and fix the notation.
- In Section 3, we demonstrate the framework of solving Avoid and Remote-Point via non-trivial algorithms for Satisfying-Pairs.
- In Section 4, we demonstrate the framework of solving Partial-Hard and Partial-AvgHard via non-trivial algorithms for Satisfying-Pairs.
- In Section 5, we present a non-trivial algorithm for ACC^0-Satisfying-Pairs from [Wil18c], and combine it with the frameworks to obtain unconditional $\mathsf{FP}^{\mathsf{NP}}$ algorithms for ACC^0-Remote-Point and ACC^0-Partial-AvgHard. We also demonstrate the consequences of these algorithms.
- In Section 6, we construct short, smooth, and rectangular PCPs of Proximity, used in the frameworks of solving Remote-Point and Partial-AvgHard.
- In Section 7, we construct short and rectangular PCPs of Proximity with query complexity only 2 or 3, used in the framework for Avoid and Partial-Hard.
- In Appendix A, we prove some technical lemmas in Section 3 and 4.

2 Preliminaries

Notation. We use $\tilde{O}(f(n))$ to denote $f(n) \cdot (\log f(n))^{O(1)}$. The concatenation of the strings x and y is denoted by $x \circ y$. The relative Hamming distance of two strings x and y, denoted by $\delta(x, y)$, is the fraction of indices i such that $x_i \neq y_i$. A string x is said to be γ-far from (resp. γ-close to) a string y if $\delta(x, y) \geq \gamma$ (resp. $\delta(x, y) < \gamma$). We say $x \in \{0,1\}^n$ is γ-far from $L \subseteq \{0,1\}^n$ if x is γ-far from every $y \in L$; otherwise x is γ-close to L. For a vector $\vec{u} \in \mathbb{R}^n$ and an integer $d \geq 1$, the ℓ_d *norm* of $\vec{u}$ is

$$\|\vec{u}\|_d := \left(\mathop{\mathbb{E}}_{i \leftarrow [n]} [|u_i|^d] \right)^{1/d}.$$

Given a function $f : \{0,1\}^n \to \{0,1\}^m$, the range of f, denoted by $\mathsf{Range}(f)$, is defined as the set of outputs of f, i.e., $\mathsf{Range}(f) := \{f(x) \mid x \in \{0,1\}^n\}$.

A function $f : \mathbb{N} \to \mathbb{N}$ is said to be *good* if there is a Turing machine such that given n in binary, it runs in time $\mathrm{poly}(\log n, \log f(n))$ and outputs $f(n)$ in binary.

A circuit class $\mathscr{C}$ is said to be *typical* if it contains the identity circuit and is closed under negations and projections. More precisely, (1) every function that always outputs its input bits is computable by a constant size $\mathscr{C}$ circuit; (2) for any $\mathscr{C}$ circuit C of size s and projection proj, both $\neg C$ and $C \circ \mathsf{proj}$ have $\mathscr{C}$ circuits of size $\mathrm{poly}(s)$, and the descriptions of these circuits can be computed in $\mathrm{poly}(s)$ time.

Let S be a finite sample space and E be an event. We use $\Pr_{x \leftarrow S}[E]$ to denote the probability that E happens if x is sampled uniformly from S. Similarly, for a random variable Y, we use $\mathbb{E}_{x \leftarrow S}[Y]$ to denote the expectation of Y when x is sampled uniformly from S.

2.1 Circuit Classes

Throughout this paper, the *size* of a circuit is defined as the number of *wires* (instead of gates) in the circuit. We will use the following (single-output) circuit classes.

- AC^0_d refers to depth-d circuits with AND and OR gates of unbounded fan-in, and NOT gates of fan-in 1. We define $\mathsf{AC}^0 := \bigcup_d \mathsf{AC}^0_d$.
- $\mathsf{AC}^0_d[m]$ refers to depth-d circuits with AND, OR, and $\mathsf{MOD}[m]$ gates of unbounded fan-in, and NOT gates of fan-in 1. A $\mathsf{MOD}[m]$ gate outputs 1 if and only if m does not divide the number of 1 in its inputs. We define $\mathsf{AC}^0[m] := \bigcup_d \mathsf{AC}^0_d[m]$. Furthermore, we define $\mathsf{ACC}^0 := \bigcup_m \mathsf{AC}^0[m]$.
- $\mathsf{CC}^0_d[m]$ refers to depth-d circuits with only $\mathsf{MOD}[m]$ gates of unbounded fan-in. We define $\mathsf{CC}^0[m] := \bigcup_d \mathsf{CC}^0_d[m]$ and $\mathsf{CC}^0 := \bigcup_m \mathsf{CC}^0[m]$.
- NC^0_d refers to constant-size circuits such that the output depends on at most d input bits. We define $\mathsf{NC}^0 := \bigcup_d \mathsf{NC}^0_d$.
- Assume that $\mathsf{F} \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}, \mathsf{MOD}[m], \dots\}$ is a gate, we define an F circuit as a circuit with only an F gate fed by some (or all) of the input bits. In particular, we define an F_d circuit as a circuit with an F gate of fan-in at most d fed by some (or all) of the input bits.

We define SYM as the class of any symmetric Boolean function, i.e., $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) = g(x_1 + x_2 + \cdots + x_n)$ for some function g.

Suppose that $\mathscr{C}_1$ and $\mathscr{C}_2$ are circuit classes, we denote $\mathscr{C}_1 \circ \mathscr{C}_2$ as the *composition* of these two classes: the input bits feed an n-input m-output $\mathscr{C}_2$ circuit C_2, and the m output bits of C_2 feed an m-input single-output $\mathscr{C}_1$ circuit C_1. For instance, a $\mathsf{SYM} \circ \mathsf{ACC}^0$ circuit contains a symmetric output gate whose inputs are ACC^0 circuits.

For a circuit class $\mathscr{C}$, we use $\mathscr{C}[s]$ to represent the sub-class of the $\mathscr{C}$ circuits of size at most s.

2.2 Error-Correcting Codes

An *error-correcting code* with *message length* n, *rate* r, and *relative distance* δ is a function $\mathsf{Enc} : \{0,1\}^n \to \{0,1\}^{rn}$ such that for every pair of distinct $x_1, x_2 \in \{0,1\}^n$, $\delta(\mathsf{Enc}(x_1), \mathsf{Enc}(x_2)) \geq \delta$. It is said to *correct* γ *fraction of errors* if there is a $\mathsf{Dec} : \{0,1\}^{rn} \to \{0,1\}$ such that for every y that is γ-close to $\mathsf{Enc}(x)$ for some $x \in \{0,1\}^n$, $\mathsf{Dec}(y) = x$.

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

Theorem 2.1 ([Spi96])**.** *There is a* $\mathrm{GF}(2)$*-linear error-correcting code* $(\mathsf{Enc}, \mathsf{Dec})$ *with a constant rate, constant relative distance, and can correct a constant fraction of errors. Moreover, both* Enc *and* Dec *are uniformly computable in linear time.*

2.3 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 $\mathsf{NTIME}[T(n)]$ to be the class of languages decidable by nondeterministic Turing machines in $T(n)$ time, and $\mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T(n), G(n)]$ to be the class of languages decidable by nondeterministic Random-Access Turing Machines (RTMs) in $T(n)$ time with $G(n)$ nondeterministic bits.

Theorem 2.2 ([FS16, CLW20, RSW22]). *Let c be a large universal constant, $T : \mathbb{N} \to \mathbb{N}$ be a good function such that $n \log^{c+1} n \le T(n) \le 2^{\mathrm{poly}(n)}$. There is a language*

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

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.*

2.4 Linear Sum Circuits and Hardness Amplification with Them

We need an XOR lemma with "linear sum" decoders: given a corrupted codeword $\tilde{f}$ that is $(1/2-\varepsilon)$-close to $\mathsf{Amp}(f)$, there is an affine transformation A such that $A(\tilde{f})$ is δ-close to f.

The actual definition of *linear sum circuits* is more involved for the following reason. Our XOR lemma works in the *list-decoding* regime, therefore it also receives an advice string α (i.e., the index in the list) and outputs the α-th decoded message in the list. When α is fixed, $A(\tilde{f};\alpha)$ is simply an affine function over $\tilde{f}$; but the dependence on α can be more complicated. It turns out that we need an upper bound on the *locality* of the dependence on α, defined as follows.

Definition 2.3 (Linear Sum Circuits)**.** Let $x \in \{0,1\}^n$ and $\alpha \in \{0,1\}^a$ be two inputs. A *linear sum* circuit on input x with advice α is a function $C : \{0,1\}^n \times \{0,1\}^a \to \mathbb{R}^m$ of the following form:

$$C(x,\alpha)_i = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha) \cdot x_{\mathsf{idx}_k(\alpha,i)}.$$

Here, A is the *fan-in* of C. The circuit is described by two functions $\mathsf{coeff}_k(\alpha)$ and $\mathsf{idx}_k(\alpha,i)$; note that $\mathsf{coeff}_k(\alpha)$ does not depend on i. For technical convenience, we will also allow $\mathsf{idx}_k(\alpha,i)$ to take special values ZERO and ONE, where x_{ZERO} is always 0 and x_{ONE} is always 1.

Besides the fan-in A, the following complexity measures of C will also be important:

- We say the *coefficient sum* of C is at most U, if for every advice α, we have

$$\sum_{k \in [A]} |\mathsf{coeff}_k(\alpha)| \le U.$$

- We say that C has *locality* l, if for every fixed k, there is a subset S_k of l bits of α such that the functions $\mathsf{coeff}_k(\alpha)$ and $\mathsf{idx}_k(\alpha,i)$ only depends on $\alpha|_{S_k}$.

Example 2.4. Consider the following example (simplified from the proof of Theorem 2.5). Suppose the advice α consists of a list of sub-advices $(\alpha_1, \alpha_2, \ldots, \alpha_{a'})$ where $a' \approx 1/\varepsilon^2$; given an index k, $\mathsf{coeff}_k(\alpha)$ only depends in α_k, and $\mathsf{idx}_k(\alpha,i)$ only depends on α_k and i. Suppose each α_k has length l, then regardless of the number a', the linear sum has locality l.

We need the following XOR lemma with linear sum decoders. The XOR lemma was proved in [Lev87, GNW11] and it was shown in [CLW20, Section A] to admit linear sum decoders. For completeness, we provide a proof of Theorem 2.5 in Appendix A.5 and verify the locality of the linear sum. Note that the XOR lemma is stated below as an approximately locally list-decodable code.

Theorem 2.5. *Let $N \in \mathbb{N}$, $0 < \varepsilon, \delta < 1/10$, $k := O(\log(1/\varepsilon)/\delta)$, $\tilde{N} := N^k$, and $a := O(\log^2 N/(\varepsilon\delta)^2)$. There is an algorithm $\mathsf{Amp} : \{0,1\}^N \to \{0,1\}^{\tilde{N}}$ computable in deterministic $\mathrm{poly}(\tilde{N})$ time, and a linear sum circuit $C : \{0,1\}^{\tilde{N}} \times \{0,1\}^a \to \mathbb{R}^N$ such that the following hold.*

(List-decoding) *For every string $\tilde{f} \in \{0,1\}^{\tilde{N}}$ that is $(1/2-\varepsilon)$-close to $\mathsf{Amp}(f)$ for some hidden string f, there is an advice $\alpha \in \{0,1\}^a$, such that (1) for every $i \in [N]$, $C(\tilde{f}, \alpha)_i \in [0,1]$; and (2) $\|C(\tilde{f}, \alpha) - f\|_1 \le \delta$.*

(Complexity) *The fan-in, coefficient sum, and locality of C are at most $O(\log N/(\varepsilon\delta)^2)$, $O(1/\varepsilon)$, and $\log \tilde{N}$ respectively.*

We will also use the notation $\mathsf{dec}_\alpha(f)$ to denote $C(f, \alpha)$, emphasising that dec_α is an affine transformation that depends on α.

2.5 PCPs of Proximity

Now we will introduce Probabilistically Checkable Proofs of Proximity (PCPPs) [BGH+06] and two properties of PCPPs that will be useful in designing algorithms for explicit construction problems: *rectangularity* and *smoothness*.

In what follows, a *pair language* is simply a subset of $\{0,1\}^* \times \{0,1\}^*$. For an instance (z, x) of a pair language, we treat z as the *explicit input* (which the PCPP verifier can read entirely) and x as the *implicit input* (which the PCPP verifier could only read a few bits). For example, Circuit-Eval is a pair language with two inputs, i.e., a circuit C and an input x, and the task is to evaluate $C(x)$. A PCPP verifier for Circuit-Eval knows the input circuit C but can only access a few bits of x.

2.5.1 Basic Definitions

Definition 2.6 (PCP of Proximity Verifiers)**.** Let $r = r(n)$, $q = q(n)$, $\ell = \ell(n)$, $d = d(n)$ be good functions and $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair language. A PCPP verifier VPCPP for L with *proof length* ℓ, *randomness complexity* r, *decision complexity* d, and *query complexity* q is a tuple of Turing machines $(V_{\mathsf{type}}, V_{\mathsf{index}}, V_{\mathsf{dec}})$ that will verify a proof $\pi \in \{0,1\}^\ell$ of the statement $(z, x) \in L \cap \{0,1\}^* \times \{0,1\}^n$ as follows.

- It randomly samples a $\mathsf{seed} \in \{0,1\}^r$ and generates

$$(\mathsf{itype}[1], \mathsf{itype}[2], \ldots, \mathsf{itype}[q]) \leftarrow V_{\mathsf{type}}(\mathsf{seed}, z),$$
$$(i[1], i[2], \ldots, i[q]) \leftarrow V_{\mathsf{index}}(\mathsf{seed}, z).$$

  For every $j \in [q]$, $\mathsf{itype}[j] \in \{\mathsf{input}, \mathsf{proof}\}$ determines the type of the j-th query: If $\mathsf{itype}[j] = \mathsf{input}$, the j-th query probes the $i[j]$-th bit of the "implicit input" x; otherwise (i.e., $\mathsf{itype}[j] = \mathsf{proof}$), the j-th query probes the $i[j]$-th bit of the proof π.

- Let $\mathsf{ans}_1, \ldots, \mathsf{ans}_q$ be the answers to the queries defined above, we say VPCPP accepts (z, x, π), denoted by $\mathsf{VPCPP}^{x \circ \Pi}(z, \mathsf{seed}) = 1$, if and only if $V_{\mathsf{dec}}(\mathsf{seed}, z, \mathsf{ans}_1, \ldots, \mathsf{ans}_q)=1$. The machine V_{dec} is said to be the *decision predicate* of VPCPP, and has circuit complexity $d(n)$.

We may represent the "implicit input" x as $\Pi_{\mathsf{input}} : [n] \to \{0,1\}$ and the proof π as $\Pi_{\mathsf{proof}} : [\ell] \to \{0,1\}$ to emphasize that they are given as oracles to VPCPP. We sometimes denote the outputs of V_{type} and V_{index} as I and denote the answers $(\mathsf{ans}_1, \ldots, \mathsf{ans}_q)$ as $(\Pi_{\mathsf{input}} \circ \Pi_{\mathsf{proof}})|_I$.

We will also consider the PCPP verifier of *pure* languages (i.e. the first part z of any input is always the empty string). In such case, we simply omit all the z in the definition above.

Definition 2.7 (PCP of Proximity)**.** Let $s = s(n)$ and $\delta = \delta(n)$ be good functions, $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair language, and $\mathsf{VPCPP} = (V_{\mathsf{type}}, V_{\mathsf{index}}, V_{\mathsf{dec}})$ be a PCPP verifier for L. We say VPCPP is a PCPP verifier for L with *completeness error* $1 - c$, *soundness error* s, and *proximity parameter* δ if the following two conditions hold for every $(z, x) \in \{0,1\}^* \times \{0,1\}^n$.

- **(Completeness).** If $(z, x) \in L$, then there is a proof $\pi \in \{0,1\}^\ell$ such that VPCPP accepts (z, x, π) with probability at least c.
- **(Soundness).** Denote $L(z)$ to be the set of $y \in \{0,1\}^n$ such that $(z, y) \in L$. If x is δ-far from $L(z)$, then for every proof $\pi \in \{0,1\}^\ell$, VPCPP accepts (z, x, π) with probability at most s.

For most of the constructions of PCPPs, the completeness error can be made 0, which means that for $(z, x) \in L$, there is a proof such that the verifier accepts with probability 1. Therefore we assume that the completeness error of a PCPP is 0 when it is not specified.

We need to define a stronger version of the soundness called *robust soundness* as follows, as an intermediate step to construct PCPPs with nice parameters.

Definition 2.8 (Robust PCP of Proximity [BGH+06])**.** Let $s = s(n)$, $\delta = \delta(n)$, and $\rho = \rho(n)$ be good functions, $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair language, and $\mathsf{VPCPP} = (V_{\mathsf{type}}, V_{\mathsf{index}}, V_{\mathsf{dec}})$ be a PCPP verifier for L. We say VPCPP is a robust PCPP verifier for L with *robust soundness error* s with robustness parameter ρ and *proximity parameter* δ if it satisfies the completeness property of PCPP and the following *robust soundness property*.

- **(Robust Soundness).** The following holds for every $(z, x) \in \{0,1\}^* \times \{0,1\}^n$. Denote $L(z)$ to be the set of $y \in \{0,1\}^n$ such that $(z, y) \in L$. If x is δ-far from $L(z)$, then for every proof $\pi \in \{0,1\}^\ell$, with probability at least $1 - s$ over the random bits seed, the answer $(\mathsf{ans}_1, \ldots, \mathsf{ans}_q)$ of the queries of VPCPP is ρ-far from being accepted (i.e. we need to flip at least a ρ fraction of the bits of the answers $(\mathsf{ans}_1, \ldots, \mathsf{ans}_q)$ to make the verifier accept).

2.5.2 Rectangular PCPs of Proximity

Following [RSW22], one of the main technical ingredients in the algorithmic method for explicit construction problems is a variant of PCPPs, called *rectangular PCPPs*. Intuitively, a rectangular PCPP verifier treats the input as an $H_{\mathsf{input}} \times W_{\mathsf{input}}$ matrix and the proof as an $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ matrix, and can generate the query indices in a "rectangular" fashion. In particular, the random seed is split into two parts denoted as $\mathsf{seed.row}$ and $\mathsf{seed.col}$ respectively, and there are two algorithms V_{row} and V_{col} such that:

- V_{row} takes $\mathsf{seed.row}$ as input and generates $\mathsf{irow}[1], \ldots, \mathsf{irow}[q]$;
- V_{col} takes $\mathsf{seed.col}$ as input and generates $\mathsf{icol}[1], \ldots, \mathsf{icol}[q]$;
- The final indices of the queries $i[1], \ldots, i[q]$ are defined as $i[j] := (\mathsf{irow}[j] - 1) \cdot W + \mathsf{icol}[j]$, where $W = W_{\mathsf{input}}$ or $W = W_{\mathsf{proof}}$ depending on the the type of the j-th query.

In other words, the row verifier V_{row} (resp. the column verifier V_{col}) takes the row randomness $\mathsf{seed.row}$ (resp. the column randomness $\mathsf{seed.col}$) and generates the row indices (resp. the column indices) of the queries. Ideally, a rectangular PCPP should satisfy the following properties:

- **(Perfect Rectangularity).** The row randomness $\mathsf{seed.row}$ and column randomness $\mathsf{seed.col}$ are independent random bits (i.e. the row and column query indices are independent).
- **(Randomness-Oblivious Type Predicate).** The type predicate V_{type}, which determines the types of the queries (i.e. whether a query is to the input or the proof oracle), does not depend on the row and column random seeds.
- **(Randomness-Oblivious Decision Predicate).** The decision predicate V_{dec}, which decides whether to accept the proof given the answers to the queries, does not depend on the row and column random seeds.

However, as in [BHPT20, RSW22], we do not know how to construct such rectangular PCPPs. Nevertheless, we could construct a weaker version where the row and column randomness are almost independent, and the dependency of the decision and type predicates on the random seeds are relatively simple. We formally define such rectangular PCPPs as follows (for simplicity, we only define rectangular PCPPs for pure languages).

Definition 2.9 (Rectangular PCPPs with Randomness-Oblivious Predicates)**.** Suppose $H_{\mathsf{input}} = H_{\mathsf{input}}(n)$, $W_{\mathsf{input}} = W_{\mathsf{input}}(n)$, $H_{\mathsf{proof}} = H_{\mathsf{proof}}(n)$, $W_{\mathsf{proof}} = W_{\mathsf{proof}}(n)$, $\tau = \tau(n)$, $p = p(n)$ are good functions such that $H_{\mathsf{input}} \cdot W_{\mathsf{input}} = O(n)$ and $L \subseteq \{0,1\}^*$ be a language. A PCPP verifier VPCPP is said to be a τ-almost rectangular PCPP that has a *randomness-oblivious predicate* (ROP) with parity-check complexity p if the following conditions hold.

- **(Randomness).** There are good functions $r_{\mathsf{row}} = r_{\mathsf{row}}(n)$, $r_{\mathsf{col}} = r_{\mathsf{col}}(n)$, and $r_{\mathsf{shared}} = r_{\mathsf{shared}}(n)$ such that the randomness complexity $r = r_{\mathsf{row}} + r_{\mathsf{col}} + r_{\mathsf{shared}}$, i.e., the random seed can be partitioned into three independent parts: the *row randomness* $\mathsf{seed.row}$, the *column randomness* $\mathsf{seed.col}$, and the *shared randomness* $\mathsf{seed.shared}$. We say r_{row}, r_{col}, and r_{shared} are the row, column, and shared randomness complexity of the PCPP verifier, respectively. Moreover, the shared randomness complexity $r_{\mathsf{shared}}(n) \leq \tau \cdot r(n)$.
- **(Query Pattern).** There are algorithms V_{type}, V_{row}, and V_{col} to generate the queries in a rectangular fashion. Concretely speaking:
  - $(\mathsf{itype}[1], \ldots, \mathsf{itype}[q]) \leftarrow V_{\mathsf{type}}(\mathsf{seed.shared})$, where $\mathsf{itype}[j] \in \{\mathsf{input}, \mathsf{proof}\}$ for all $j \in [q]$.
  - $(\mathsf{irow}[1], \ldots, \mathsf{irow}[q]) \leftarrow V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared})$
  - $(\mathsf{icol}[1], \ldots, \mathsf{icol}[q]) \leftarrow V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared})$
  - For every $j \in [q]$, the index of the j-th query $i[j] := \mathsf{irow}[j] \cdot W + \mathsf{icol}[j]$, where $W = W_{\mathsf{input}}$ if $\mathsf{itype}[j] = \mathsf{input}$ and $W = W_{\mathsf{proof}}$ otherwise.

  As normal PCPP verifiers, the j-th query is to the $i[j]$-th bit of the input if $\mathsf{itype}[j] = \mathsf{input}$, and is to the $i[j]$-th bit of the proof if $\mathsf{itype}[j] = \mathsf{proof}$. Note that since $H_{\mathsf{input}} \cdot W_{\mathsf{input}}$ may be larger than n, the query to the input is not well-defined when $i[j] > n$. In such case, we denote the answer to be $\perp$.
- **(Decision Predicate).** There are polynomial-time algorithms V_{dec} and V_{pc} such that the following holds.
  - The algorithm $V_{\mathsf{dec}}(\mathsf{seed.shared})$ generates a circuit $\mathsf{VDec} : \{0,1,\perp\}^{p+q} \to \{0,1\}$.
  - The algorithm $V_{\mathsf{pc}}(\mathsf{seed.shared})$ generates p XOR circuits (i.e. circuits computing GF(2)-linear functions) $pc_1, \ldots, pc_n : \{0,1\}^{r_{\mathsf{row}}+r_{\mathsf{col}}} \to \{0,1\}$.

Assume that $(\mathsf{ans}_1, \ldots, \mathsf{ans}_q) \in \{0,1,\bot\}^q$ are the answers to the queries. For every $i \in [p]$, we denote $pc_i := pc_i(\mathsf{seed.row}, \mathsf{seed.col})$. The PCPP verifier accepts the proof if

$$\mathsf{VDec}(\mathsf{ans}_1, \ldots, \mathsf{ans}_q, pc_1, \ldots, pc_p) = 1.$$

The *decision complexity* of this PCPP verifier is said to be the circuit complexity of V_{dec}.

2.5.3 Smooth PCP of Proximity

Apart from rectangularity, another important property of the PCPPs is their *smoothness*. Roughly speaking, it means that the queries to the proof oracle are smooth, in the sense that each location is probed with equal probability.[12] The formal definition is as follows.

Definition 2.10 (Smooth PCPPs for Pure Languages)**.** Let $r = r(n)$, $q = q(n)$ be good functions, $L \subseteq \{0,1\}^*$ be a language, and $\mathsf{VPCPP} = (V_{\mathsf{type}}, V_{\mathsf{index}}, V_{\mathsf{dec}})$ be a PCPP verifier for L with randomness complexity r. It is said to be a *smooth PCPP verifier* if for all locations $\mathsf{loc}_1, \mathsf{loc}_2$ in the proof oracle, over a uniformly random $\mathsf{seed} \in \{0,1\}^r$ and a uniformly random index $j \in [q]$, loc_1 and loc_2 are probed by VPCPP with equal probability in the j-th query.

Smooth PCPPs can be viewed as PCPPs that can tolerate errors in the proof: since all the locations in the proof are queried with equal probability, a slightly corrupted version of a correct proof is still likely to be accepted, as shown in the following lemma.

Lemma 2.11. *Let $q = q(n)$, $\ell = \ell(n)$, $s = s(n)$ be good functions, $L \subseteq \{0,1\}^*$ be a language, and VPCPP be a smooth PCPP verifier for L with soundness error s, proof length ℓ, and query complexity q. Assume that $x \in L \cap \{0,1\}^n$ and $\pi \in \{0,1\}^\ell$ is a correct proof for $x \in L$, i.e., $\mathsf{VPCPP}^{x \circ \pi}(\mathsf{seed})$ accepts with probability 1 over $\mathsf{seed} \leftarrow \{0,1\}^r$. Then for every π' such that the relative Hamming distance between π' and π is at most ε, $\mathsf{VPCPP}^{x \circ \pi'}(\mathsf{seed})$ accepts with probability at least $1 - q \cdot \varepsilon$ over $\mathsf{seed} \sim \{0,1\}^r$.*

Proof. We say a location $i \in [\ell]$ of the proof oracle is *bad* if $\pi[i] \neq \pi'[i]$. Let B_j be the event that the j-th query of VPCPP probes a bad location in the proof. By the smoothness, we know that

$$\Pr_{\mathsf{seed} \in \{0,1\}^r, j \in [q]} \left[B_j \right] \leq \varepsilon.$$

By a union bound, we can see that

$$\Pr_{\mathsf{seed} \in \{0,1\}^r} \left[\exists\ j \in [q],\ B_j \right] \leq \sum_{j \in [q]} \Pr_{\mathsf{seed} \in \{0,1\}^r} \left[B_j \right] \leq q \cdot \varepsilon. \tag{2}$$

Denote E be the event that there exists a $j \in [q]$ such that B_j happens. Then it follows that

$$\begin{aligned}
& \Pr_{\mathsf{seed} \in \{0,1\}^r} \left[\mathsf{VPCPP}^{x \circ \pi'}(\mathsf{seed}) \text{ rejects} \right] \\
\leq\ & \Pr_{\mathsf{seed} \in \{0,1\}^r} \left[\mathsf{VPCPP}^{x \circ \pi'}(\mathsf{seed}) \text{ rejects} \,\middle|\, \neg E \right] + \Pr_{\mathsf{seed} \in \{0,1\}^r} \left[E \right] \\
\leq\ & 0 + q \cdot e \\
=\ & q \cdot e,
\end{aligned}$$

where the second inequality follows from Equation (2) and the perfect completeness of VPCPP. □

[12] In some literature (e.g. [Par21]), the smoothness of PCPPs is defined differently: the queries to both the input oracle and the proof oracle need to be smooth, i.e., each location in the input (resp. the proof) is queried with equal probability. Here, we only require the queries to the proof oracle to be smooth and have no requirement on the query distribution over the input oracle.

Note that smoothness can be defined for rectangular PCPPs, in which case each location in the proof matrix is probed with equal probability. This further means that each row (resp. column) index is queried by the row (resp. column) verifier with equal probability.

Remark 2.12. A stronger definition of the smoothness is as follows: for every fixed $i \in [q]$, condition on the i-th query probing the proof oracle, the i-th query is uniformly random over the proof oracle. By randomly permuting the q queries, we can make a smooth PCPP satisfy this stronger definition of smoothness. In particular, if we have a smooth and rectangularity PCPP, we can make it satisfy this stronger definition of smoothness by adding $O(q \log q)$ bits in the shared randomness for a random permutation over the q queries.

2.5.4 Our Constructions

In this work, we will need two new constructions of rectangular PCPPs: for the range avoidance problem and worst-case hard partial truth tables, we need a rectangular PCPP with query complexity 3 or 2 (depending whether perfect completeness is required); for the remote point problem and average-case hard partial truth table, we need a smooth and rectangular PCPP with query complexity $O(1)$.[13]

Theorem 2.13 (3-Query and 2-Query Rectangular PCPPs)**.** *For every constant $\delta \in (0,1)$, there are constants $s_3 \in (0,1)$ and $0 < s_2 < c_2 < 1$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \le m \le (\log T(n))^{0.1}$, $n \le T(n) \le 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \le \log T(n)$, and $w_{\mathsf{input}}(n) \le \log n$. Then there are good functions $h^{\mathsf{3q}}_{\mathsf{proof}}(n)$, $h^{\mathsf{2q}}_{\mathsf{proof}}(n)$, and $h_{\mathsf{input}}(n)$ satisfying*

$$h^{\mathsf{3q}}_{\mathsf{proof}}(n), h^{\mathsf{2q}}_{\mathsf{proof}}(n) = \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}(n), \qquad \text{and}$$
$$h_{\mathsf{input}}(n) = \lceil \log n \rceil - w_{\mathsf{input}}(n),$$

such that the following holds.

Suppose that $w_{\mathsf{proof}}, h^{\mathsf{3q}}_{\mathsf{proof}}, h^{\mathsf{2q}}_{\mathsf{proof}} \ge (5/m) \log T(n)$, and that for some absolute constant $C \ge 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h^{\mathsf{3q}}_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h^{\mathsf{2q}}_{\mathsf{proof}}(n)} \le 1 - \frac{Cm \log\log T(n)}{\log T(n)}.$$

Let $W_{\mathsf{proof}}(n) := 2^{w_{\mathsf{proof}}(n)}$, $H^{\mathsf{3q}}_{\mathsf{proof}}(n) := 2^{h^{\mathsf{3q}}_{\mathsf{proof}}(n)}$, $H^{\mathsf{2q}}_{\mathsf{proof}}(n) := 2^{h^{\mathsf{2q}}_{\mathsf{proof}}(n)}$, $W_{\mathsf{input}}(n) := 2^{w_{\mathsf{input}}(n)}$, and $H_{\mathsf{input}}(n) := 2^{h_{\mathsf{input}}(n)}$. Then $\mathsf{NTIME}[T(n)]$ has:

- *a rectangular PCP of proximity V_{3q} with perfect completeness, soundness error s_3, an $H^{\mathsf{3q}}_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix and an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix;*
- *a rectangular PCP of proximity V_{2q} with completeness error $1 - c_2$, soundness error s_2, an $H^{\mathsf{2q}}_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix and an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix.*

Other parameters of V_{3q} and V_{2q} are specified in Table 1.

Furthermore, given the randomness $\mathsf{seed} \in \{0,1\}^r$, the total number of queries and parity-check bits is at most 3 for V^{3q} and 2 for V^{2q}, and the decision predicate $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ of the rectangular PCPP verifier is an OR of the input bits (including queries and parity-check bits) or their negations for every $\mathsf{seed.shared}$.

[13] Unfortunately, our smooth PCPP requires a large (although constant) number of queries, because of the arguments in Section 6.1.

PCPP Verifier	$V^{3\mathsf{q}}$	$V^{2\mathsf{q}}$
Completeness error	0	$1-c_2$
Soundness error	s_3	s_2
Proximity parameter	δ	
Row randomness	$h^{3\mathsf{q}}_{\mathsf{proof}} - (5/m)\log T(n)$	$h^{2\mathsf{q}}_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (5/m)\log T(n)$	
Shared randomness	$(10/m)\log T(n) + O(\log\log T(n) + m\log m)$	
Query complexity	3	2
Parity check complexity		
Decision complexity	$\mathrm{poly}(\log\log T)$	

Table 1: Parameters of the PCPPs constructed in Theorem 2.13.

Theorem 2.14 (Smooth and Rectangular PCPP)**.** *For all constants $\delta \in (0,1)$ and $s \in (0,1)$, there is a constant $q \geq 1$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \leq m(n) \leq (\log T(n))^{0.1}$, $n \leq T(n) \leq 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \leq \log T(n)$, and $w_{\mathsf{input}}(n) \leq \log n$. Then there are good functions $h_{\mathsf{proof}}(n)$ and $h_{\mathsf{input}}(n)$ satisfying*

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

such that the following holds.

Suppose that $w_{\mathsf{proof}}, h_{\mathsf{proof}} \geq (5/m)\log T(n)$, and that for some absolute constant $C \geq 1$,

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \leq 1 - \frac{Cm^2\log\log T(n)}{\log T(n)}.$$

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 smooth and rectangular PCP of proximity with an $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ input matrix and an $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ proof matrix, whose other parameters are specified in Table 2.

Soundness error	s
Proximity parameter	δ
Row randomness	$r_{\mathsf{row}} := h_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$r_{\mathsf{col}} := w_{\mathsf{proof}} - (5/m)\log T(n)$
Shared randomness	$r_{\mathsf{shared}} := (10/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	$q = O_{s,\delta}(1)$
Parity check complexity	
Decision complexity	$\mathrm{poly}(T(n)^{1/m})$

Table 2: Parameters of the PCPP constructed in Theorem 2.14.

Details of these two constructions are postponed to Section 6 and Section 7.

2.6 A Stretch Reduction for Remote-Point and Partial-AvgHard

In our framework for solving Remote-Point (Section 3.2), for technical convenience, we only consider circuits $C : \{0,1\}^n \to \{0,1\}^{\ell(n)}$, where $\ell(n)$ is a certain stretch function. (For example, it

might be the case that $\ell(n)$ is rounded to a power of 2 for every n.) In this subsection, we show that such an algorithm can also solve Remote-Point for circuits of larger stretches. This justifies that it is without loss of generality to only consider stretch functions that are *equal* to $\ell(n)$.

Lemma 2.15 (Stretch Reduction for Remote-Point)**.** *Let $\mathscr{C}$ be a typical circuit class and s be a size parameter. Suppose that $\mathscr{C}[s]$-Remote-Point with stretch $\ell'(n)$ and distance parameter $1/2 - \varepsilon'(n)$ admits an $\mathsf{FP}^{\mathsf{NP}}$ algorithm. Then for any stretch $\ell = \ell(n) \geq \ell'(n+1)/2$, $\mathscr{C}[s]$-Remote-Point with stretch $\ell(n)$ and distance parameter $1/2 - \varepsilon(n)$ also admits an $\mathsf{FP}^{\mathsf{NP}}$ algorithm, where $\varepsilon(n) := 2 \cdot \varepsilon'(n+1)$.*

Proof. Denote $\ell' := \ell'(n+1)$, $\varepsilon' := \varepsilon'(n+1)$, $\ell := \ell(n)$, and $\varepsilon := \varepsilon(n)$, and let $C : \{0,1\}^n \to \{0,1\}^\ell$ be an input circuit. If ℓ is a multiple of ℓ', we can split the ℓ-bit output of C into blocks of size ℓ' and add a dummy input bit to construct $m := \ell/\ell'$ circuits $C_1, C_2, \ldots, C_m : \{0,1\}^{n+1} \to \{0,1\}^{\ell'(n+1)}$ such that for every $x \in \{0,1\}^n$ and $b \in \{0,1\}$,

$$C(x) = C_1(x,b) \circ C_2(x,b) \circ \cdots \circ C_m(x,b).$$

Using the $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Remote-Point with stretch $\ell'(n+1)$ and error parameter $\varepsilon'(n+1)$, we can construct $y_1, y_2, \ldots, y_m \in \{0,1\}^{\ell'}$ such that each y_i is $(1/2 - \varepsilon')$-far from $\mathsf{Range}(C_i)$. It then follows that the concatenation $y_1 \circ y_2 \circ \cdots \circ y_m$ is $(1/2 - \varepsilon')$-far from $\mathsf{Range}(C)$.

We now consider the case where ℓ is not a multiple of ℓ'. Let $I : \{0,1\}^{n+1} \to \{0,1\}$ be defined as the projection $I(x) = x_{n+1}$, that is, it always outputs the last bit. For any t, let $I^{\otimes t} : \{0,1\}^{n+1} \to \{0,1\}^t$ denote the concatenation of t copies of I. Therefore, $\mathsf{Range}(I^{\otimes t}) = \{0^t, 1^t\}$. Since $\mathscr{C}$ is typical, we have $I^{\otimes t} \in \mathscr{C}$.

Let $M = k \cdot \ell'$ be the nearest multiple of ℓ' larger than ℓ, and $\bar{\ell} := M - \ell$. For a multi-output $\mathscr{C}$ circuit C, we define $\tilde{C} : \{0,1\}^{n+1} \to \{0,1\}^M$ as

$$\tilde{C}(x,b) = C(x) \circ I^{\otimes \bar{\ell}}(x,b),$$

where $x \in \{0,1\}^n$ and $b \in \{0,1\}$. Since $\tilde{C}$ is of input length $n+1$ and output length being a multiple of ℓ', we can get a remote point $s \in \{0,1\}^M$ in $\mathsf{FP}^{\mathsf{NP}}$ that is $(1/2 - \varepsilon')$-far from $\mathsf{Range}(\tilde{C})$.

Let $s = s_1 \circ s_2$, where s_1 and s_2 has length ℓ and $\bar{\ell}$, respectively. We then prove that s_1 is $(1/2 - \varepsilon)$ far from $\mathsf{Range}(C)$. Towards a contradiction, we assume that s_1 is not $(1/2 - \varepsilon)$-far from $\mathsf{Range}(C)$. In other words, there is an $x \in \{0,1\}^n$ such that $\delta(C(x), s_1) < 1/2 - \varepsilon$. By considering the Hamming weight of s_2 we know that there is a $b \in \{0,1\}$ such that $\delta(I^{\otimes \bar{\ell}}(x,b), s_2) \leq 1/2$. It then follows that

$$\begin{aligned}
\delta(s, \tilde{C}(x,b)) &= \delta(s_1 \circ s_2, C(x) \circ I^{\otimes \bar{\ell}}(x,b))) \\
&\leq \frac{\ell}{\ell + \bar{\ell}} \cdot \left(\frac{1}{2} - \varepsilon(n)\right) + \frac{\bar{\ell}}{\ell + \bar{\ell}} \cdot \frac{1}{2} \\
&\leq \frac{1}{2} - \frac{\ell}{\ell + \bar{\ell}} \cdot \varepsilon(n) \\
&< \frac{1}{2} - \varepsilon'(n+1).
\end{aligned}$$

This leads to a contradiction as s is $(1/2 - \varepsilon'(n+1))$-far from $\mathsf{Range}(\tilde{C})$. □

Similar to Remote-Point, another average-case problem Partial-AvgHard can also be reduced to the instances with smaller stretch in the same way.

Lemma 2.16 (Stretch Reduction for Partial-AvgHard)**.** *Let $\mathscr{C}$ be a typical circuit class and s be a size parameter. Suppose that $\mathsf{NC}^0_2 \circ (\mathscr{C}[s])$-Partial-AvgHard with stretch $\ell'(n)$ and the distance parameter $1/2-\varepsilon'(n)$ admits an $\mathsf{FP}^{\mathsf{NP}}$ algorithm, then for any stretch $\ell = \ell(n) \geq \ell'(n+1)/2$, $\mathscr{C}[s]$-Partial-AvgHard with stretch $\ell(n)$ and distance parameter $1/2-\varepsilon(n)$ admits an $\mathsf{FP}^{\mathsf{NP}}$ algorithm, where $\varepsilon(n) := 2 \cdot \varepsilon'(n+1)$.*

Proof Sketch. The proof of this lemma is similar to that of Lemma 2.15. Here we use the same notation as the proof of Lemma 2.15.

Let $X = \{x_1, \ldots, x_\ell\}$ denote input strings, and let $y_i := x_i \circ 0$. We create $\bar{\ell}$ copies of $0^n \circ 1$ and use $y_{\ell+1}, \ldots, y_M$ to denote these copies.

we solve $\mathsf{NC}^0_2 \circ \mathscr{C}$-Partial-AvgHard on $\{y_1, \ldots, y_M\}$ and get an average-case hard partial truth table $s = s_1 \circ s_2$ that is $(1/2 - \varepsilon'(n+1))$-far from any truth table of $\mathsf{NC}^0_2 \circ \mathscr{C}$ circuit, where s_1 and s_2 has length ℓ and $\bar{\ell}$. Then we prove s_1 is a solution for the original problem. For some $\mathscr{C}$ circuit C, if s_1 is not $(1/2-\varepsilon(n))$ far from partial truth table of C on X, we can define $\tilde{C}_1, \tilde{C}_2 : \{0,1\}^n \times \{0,1\} \to \{0,1\}$ as $\tilde{C}_1(x;b) := C(x) \vee b, \tilde{C}_2(x;b) := C(x) \wedge (\neg b)$. Then one of $\tilde{C}_1$ and $\tilde{C}_2$ has partial truth table on $Y := \{y_1, \ldots, y_M\}$ not $(1/2 - \varepsilon'(n+1))$ far from s, which leads to a contradiction. Therefore s_1 has to be a solution. The analysis is similar to Lemma 2.15. □

2.7 Satisfying Pairs for $\mathsf{NC}^0_d \circ \mathscr{C}$ from Satisfying Pairs for $(\mathsf{AND}^0_d/\mathsf{XOR}^0_d/\mathsf{OR}^0_d) \circ \mathscr{C}$

We show that satisfying pairs for $\mathsf{NC}^0_d \circ \mathscr{C}$ circuits can be reduced to the satisfying pairs of $\mathsf{AND}^0_d \circ \mathscr{C}$, $\mathsf{XOR}^0_d \circ \mathscr{C}$, or $\mathsf{OR}^0_d \circ \mathscr{C}$ via standard Fourier analysis (see, e.g., [CW19b, Section 4]). This will be beneficial for the unconditional results for weak circuit classes, such as the remote point algorithm for GF(2)-linear functions.

Theorem 2.17. *For every constants $\delta \in [0,1]$ and $d \geq 1$, there is a constant δ' such that the following holds. Let $N = N(n), M = M(n), n, s = s(n)$ be parameters, and $\mathsf{C}_d \in \{\mathsf{AND}_d, \mathsf{OR}_d, \mathsf{XOR}_d\}$.*

Then $\#(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs (resp. Approx_δ-$(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs) with parameters (N, M, n, s) is $\tilde{O}(n)$-time Turing-reducible to $\#(\mathsf{C}_d \circ \mathscr{C})$-Satisfying-Pairs (resp. $\mathsf{Approx}_{\delta'}$-$(\mathsf{C}_d \circ \mathscr{C})$-Satisfying-Pairs) with parameters $(\Theta(N), M, n, s)$, as long as each input circuit of the $\#(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs (resp. Approx_δ-$(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs) problem are given explicitly as a top NC^0_d circuit C_{top} together with d circuits $C_1, C_2, \ldots, C_d \in \mathscr{C}$ feeding C_{top}.

Moreover, the oracle algorithm for $\#(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs (resp. Approx_δ-$(\mathsf{NC}^0_d \circ \mathscr{C})$-Satisfying-Pairs) only makes $O(1)$ non-adaptive queries to the $\#(\mathsf{C}_d \circ \mathscr{C})$-Satisfying-Pairs (resp. $\mathsf{Approx}_{\delta'}$-$(\mathsf{C}_d \circ \mathscr{C})$-Satisfying-Pairs) oracle.

Proof. Let N, M, n, s be the parameters. Suppose that we are given $C_1, C_2, \ldots, C_N \in \mathscr{C}[s]$ and $x_1, x_2, \ldots, x_M \in \{0,1\}^n$ as input. We assume that $C_1, C_2, \ldots, C_N$ share the same upper NC^0_d function computing $f : \{0,1\}^d \to \{0,1\}$, that is for every $i \in [N]$, $C_i \equiv f \circ D_i$ for some d-output $\mathscr{C}$ circuit D_i of size at most s. This is without loss of generality since there are at most $2^{2^d} = O(1)$ different NC^0_d functions and we can (approximately) count the number of satisfying pairs for each of these cases separately.

We first consider the case for $\mathsf{C}_d = \mathsf{AND}_d$. We use the basis $\{0,1\} \subseteq \mathbb{R}$ for Boolean values and write f as

$$f(x) = \sum_{S \subseteq [d]} \alpha_S \cdot \prod_{i \in S} x_i,$$

where each coefficient $\alpha_S \in [-2^d, 2^d] \cap \mathbb{Z}$. Note that we can compute the coefficients by writing the truth table of f in the canonical disjunctive normal form, represent x by x, $\neg x$ by $1-x$, $\wedge$ by

multiplication, and (disjoint) $\vee$ by addition, and then expending the multi-linear polynomial using a brute-force algorithm in $O(1)$ time.

Let $\chi_S(x) := \prod_{i \in S} x_i$ for $S \subseteq [d]$. Then the number of $(i,j) \in [N] \times [M]$ such that $C_i(x_j) = 1$ is

$$\begin{aligned}
&\sum_{i\in[N]} \sum_{j\in[M]} f(D_i(x_j)) \\
&= \sum_{i\in[N]} \sum_{j\in[M]} \sum_{S\subseteq[d]} \alpha_S \cdot \chi_S(D_i(x_j)) \\
&= \sum_{S\subseteq[d]} \alpha_S \cdot \left[\sum_{i\in[N]} \sum_{j\in[M]} \chi_S(D_i(x_j)) \right] \\
&= \sum_{S\subseteq[d]} \alpha_S \cdot \left[\sum_{i\in[N]} \sum_{j\in[M]} \mathsf{AND}_{|S|} \circ D_i|_S(x_j) \right],
\end{aligned}$$

where $D_i|_S : \{0,1\}^{|S|} \to \{0,1\}$ representing the circuit obtained from D_i by restricting to the output bits in S. Then our algorithm is as follows: We enumerate all $S \subseteq [d]$ and count (resp. approximately count) the number A_S of satisfying pairs for circuits $\mathsf{AND}_{|S|} \circ D_1|_S, \ldots, \mathsf{AND}_{|S|} \circ D_N|_S$ and inputs $x_1, \ldots, x_M$, then we output the answer $\sum_{S\subseteq[d]} \alpha_S \cdot A_S$.

For $\mathsf{C}_d = \mathsf{XOR}_d$ and $\mathsf{C}_d = \mathsf{OR}_d$, we only need to write f as

$$f(x) = \sum_{S\subseteq[d]} \alpha'_S \cdot \bigoplus_{i\in S} x_i, \tag{3}$$

$$f(x) = \sum_{S\subseteq[d]} \alpha''_S \cdot \bigvee_{i\in S} x_i, \tag{4}$$

where $\alpha'_S, \alpha''_S \leq 2^{O(d)}$. Note that Equation (3) can be obtained using the basis $\{\mathsf{true} := -1, \mathsf{false} := 1\}$ and Equation (4) can be obtained using the basis $\{\mathsf{true} := 0, \mathsf{false} := 1\}$. □

3 Range Avoidance and Remote Point

In this section, we prove our main technical result: non-trivial algorithms for Satisfying-Pairs imply $\mathsf{FP}^{\mathsf{NP}}$ algorithms for range avoidance and remote point. We first prove our results for range avoidance and remote point in Section 3.1 and Section 3.2, respectively, and then briefly discuss the variants of our algorithms in Section 3.3.

The naïve algorithm for Satisfying-Pairs is to evaluate every circuit on every input, which requires $O(NM \cdot \mathrm{poly}(s,n))$ time. We will employ non-trivial algorithms (i.e. of time complexity $NM/\log^{\omega(1)}(NM)$) for Satisfying-Pairs to solve the range avoidance problem and the remote point problem. Indeed, we can even allow the Satisfying Pairs algorithm to have a *preprocessing phase*, in which a polynomial-time algorithm with access to an NP oracle is given the circuits $C_1, \ldots, C_N$ (but not the inputs) and produce a data structure of small (i.e. "fixed polynomial") size.

Definition 3.1 (Algorithms for Satisfying-Pairs with P^{NP} Preprocessing on Circuits)**.** Let P be one of the problems $\mathscr{C}$-Satisfying-Pairs, $\#\mathscr{C}$-Satisfying-Pairs, Approx_δ-$\mathscr{C}$-Satisfying-Pairs, Gap_δ-$\mathscr{C}$-Satisfying-Pairs. A t-time algorithm for P with P^{NP} preprocessing of an ℓ-size data structure on circuits is a pair of algorithms (A_1, A_2) that solves P in two phases:

1. Given the circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s, the polynomial-time algorithm A_1 with oracle access to a SAT oracle computes a string $\mathsf{DS} \in \{0,1\}^\ell$.
2. Given the inputs $x_1, x_2, \ldots, x_M \in \{0,1\}^n$ and the string $\mathsf{DS} \in \{0,1\}^\ell$, the algorithm A_2 solves P on the instance $(C_1, \ldots, C_N, x_1, \ldots, x_M)$ in time t.

3.1 Range Avoidance from Satisfying-Pairs

In this sub-section, we establish the connection between the Avoid and Satisfying-Pairs. The main result is the following theorem.

Theorem 3.2. *There are constants $\varepsilon > 0$ and c_{imp} such that the following holds. Let $0 < \eta < 1/2$ be a constant, $\ell(n) > n^{1+4\eta}$ be a good function. Let $\mathscr{C}[s]$ be a typical circuit class where $s = s(n)$ is a size parameter, and $\mathscr{C}'[2s] := \mathsf{OR}_2 \circ \mathscr{C}[s]$ (i.e. a $\mathscr{C}'$ circuit of size $2s$ refers to the OR of at most two $\mathscr{C}$ circuits of size s).*

Assumption: *Suppose that for some constant $c \geq 1$, there is an $(NM/\log^{c_{\mathsf{imp}}}(NM))$-time algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs with $N := \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$ circuits of size $2s(n)$ and $M := \ell^{c+1-\eta} \cdot \mathrm{polylog}(\ell)$ inputs of length $n \cdot \mathrm{polylog}(\ell)$, allowing a P^{NP} preprocessing of an N^c-size data structure on circuits.*

Conclusion: *Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Avoid with stretch $\ell(n)$.*

3.1.1 Typical Choices of the Parameters

Before proving this general framework, we demonstrate two typical choices of the parameters as follows that deal with Avoid with polynomial stretch and quasi-polynomial stretch.

Corollary 1.11. *There is a constant $\varepsilon > 0$ such that the following holds. Let $\mathscr{C}$ be a typical circuit class, $\mathscr{C}' := \mathsf{OR}_2 \circ \mathscr{C}$, and $s = s(n)$ be a non-decreasing size parameter.*

- *Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for $N = n^{1+\Omega(1)}$ $\mathscr{C}'$-circuits of size $2s(n)$ and $M = n^{1+\Omega(1)}$ inputs of length n. Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid with stretch ℓ and circuit size s,[14] for some $\ell = n^{1+\Omega(1)}$.*
- *Suppose that there is a non-trivial algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for $N = \text{quasi-poly}(n)$ $\mathscr{C}'$-circuits of size $2s(n)$ and $M = \text{quasi-poly}(n)$ inputs of length n. Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Avoid with stretch ℓ and circuit size s, for some $\ell = \text{quasi-poly}(n)$.*

Proof. (*Polynomial Case*). Assume we have an algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs for $N_{\mathsf{al}} = n_{\mathsf{al}}^{1+c_n}$ $\mathscr{C}'$ circuits of size $2s_{\mathsf{al}}(n_{\mathsf{al}})$ and $M_{\mathsf{al}} = n_{\mathsf{al}}^{1+c_m}$ inputs of length n_{al} that runs in non-trivial time (i.e. $N_{\mathsf{al}} \cdot M_{\mathsf{al}}/\log^{\omega(1)}(N_{\mathsf{al}} \cdot M_{\mathsf{al}})$), where c_n and c_m are some constants. Denote this algorithm by $(*)$. Note that $s(n) = \mathrm{poly}(n)$ (otherwise there cannot be such an algorithm). We apply Theorem 3.2 with $\eta = 0.1$ and $s := s_{\mathsf{al}}$. Let $c = 1$ be the constant in Theorem 3.2 and $\ell(n) = \mathrm{poly}(n)$ to be determined later.

To obtain an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Avoid with stretch $\ell(n)$, we need to design an algorithm for $\mathscr{C}'$-Satisfying-Pairs with $N = \ell^{0.9} \cdot \mathrm{polylog}(\ell)$ circuits of size $s(n_{\mathsf{fw}})$ and $M = \ell^{1.9} \cdot \mathrm{polylog}(\ell)$ inputs of length $n_{\mathsf{fw}} = n \cdot \mathrm{polylog}(\ell)$ that runs in time $NM/\log^{c_{\mathsf{imp}}}(NM)$ for some absolute constant c_{imp}. Denote this problem by $(\star)$. We set $n_{\mathsf{al}} = n_{\mathsf{fw}}$ and $\ell(n) = \max\{(N_{\mathsf{al}} s)^{1.1/0.9}, (M_{\mathsf{al}} s)^{1.1/1.9}\} = n^{1+\Omega(1)}$, so that $N \geq N_{\mathsf{al}}^{1.1} \cdot s^{1.1}$ and $M \geq M_{\mathsf{al}}^{1.1} \cdot s^{1.1}$. The algorithm for $(\star)$ works as follows.

[14] Note that the circuit size parameter of $\mathscr{C}$-Avoid refers to the maximum circuit size of each output bit of $C : \{0,1\}^n \to \{0,1\}^\ell$, instead of the total circuit size of C.

- We divide the N circuits into groups of size N_{al} and the M inputs into groups of size M_{al}.
- For every $i \in [N/N_{\mathsf{al}}]$ and $j \in [M/M_{\mathsf{al}}]$, we use the algorithm $(*)$ to (approximately) count the number of satisfying pairs between the i-th circuit group and the j-th input group. (In particular, we use a brute-force algorithm with running time $(N_{\mathsf{al}} \cdot M + M_{\mathsf{al}} \cdot N) \cdot \tilde{O}(s) \le NM/\log^{\omega(1)}(NM)$ to deal with the groups, if any, with less than N_{al} circuits or less than M_{al} inputs.) We add all the answers among groups together.

The correctness of the algorithm is obvious, and the running time is at most

$$\frac{N}{N_{\mathsf{al}}} \cdot \frac{M}{M_{\mathsf{al}}} \cdot \frac{N_{\mathsf{al}} M_{\mathsf{al}}}{\log^{\omega(1)}(N_{\mathsf{al}} M_{\mathsf{al}})} \le \frac{NM}{\log^{c_{\mathsf{imp}}}(NM)}.$$

(*Quasi-Polynomial Case.*) The proof is almost the same as the polynomial case. In particular, $s(n)$ and $\ell(n)$ as defined above are quasi-polynomial functions; the final running time is non-trivial since $\log(N_{\mathsf{al}} M_{\mathsf{al}}) = \log^{\Omega(1)}(n)$ and $\log(NM) = \log^{O(1)}(n)$. The details are omitted. □

3.1.2 Proof of Theorem 3.2

Theorem 3.2. *There are constants $\varepsilon > 0$ and c_{imp} such that the following holds. Let $0 < \eta < 1/2$ be a constant, $\ell(n) > n^{1+4\eta}$ be a good function. Let $\mathscr{C}[s]$ be a typical circuit class where $s = s(n)$ is a size parameter, and $\mathscr{C}'[2s] := \mathsf{OR}_2 \circ \mathscr{C}[s]$ (i.e. a $\mathscr{C}'$ circuit of size $2s$ refers to the OR of at most two $\mathscr{C}$ circuits of size s).*

Assumption: *Suppose that for some constant $c \ge 1$, there is an $(NM/\log^{c_{\mathsf{imp}}}(NM))$-time algorithm for $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs with $N := \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$ circuits of size $2s(n)$ and $M := \ell^{c+1-\eta} \cdot \mathrm{polylog}(\ell)$ inputs of length $n \cdot \mathrm{polylog}(\ell)$, allowing a P^{NP} preprocessing of an N^c-size data structure on circuits.*

Conclusion: *Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Avoid with stretch $\ell(n)$.*

Proof. Suppose that we are given a $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}^\ell$. Without loss of generality, we may assume ℓ is a power of 2 and $c \ge 2$. We set the following parameters:

$$\begin{aligned} m &:= 5(c+2)/\eta = O(1), \\ w_{\mathsf{proof}} &:= \log \ell, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = \ell, \\ h_{\mathsf{proof}} &:= (c+1)\log \ell, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = \ell^{c+1}, \\ n_{\mathsf{hard}} &:= 10 H_{\mathsf{proof}} \cdot n, \\ T &:= H_{\mathsf{proof}} \cdot W_{\mathsf{proof}} / \log^{c_{\mathsf{tm}}}(\ell). \end{aligned}$$

The constants ε, c_{imp}, and c_{tm} will be determined later.

Let L^{hard} be the hard language constructed in Theorem 2.2. We use n_{hard} and T to denote the input length and the time complexity of L^{hard}, respectively, i.e.

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

where c_{hard} is some large universal constant. Note that since $T = \ell^{c+2}/\mathrm{polylog}(\ell)$, $n_{\mathsf{hard}} = O(\ell^{c+1} \cdot n)$, $\ell > n^{1+4\eta}$, we can see that $n_{\mathsf{hard}}^{1+\Omega(1)} \le T \le n_{\mathsf{hard}}^2$, which satisfies the technical condition of Theorem 2.2.

We describe a nondeterministic RAM M^{PCPP} that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, uses $n_{\mathsf{hard}}/10$ advice bits, guesses $n_{\mathsf{hard}}/10$ nondeterministic bits, and attempts to solve L^{hard} on n_{hard}-bit inputs.

By the definition of L^{hard}, M^{PCPP} has to fail on some input $x \in \{0,1\}^{n_{\mathsf{hard}}}$ when n_{hard} is sufficiently large. Our goal is to design such an algorithm M^{PCPP} that (1) rejects every $x \notin L^{\mathsf{hard}}$, and (2) accepts every $x \in L^{\mathsf{hard}}$ *with an easy witness*. Thus, if M^{PCPP} fails on some input x, then $x \in L$ and it has only "hard witnesses", which will be exploited for finding a non-output of C.

Here, to define the inputs x "with an easy witness", we will need the 2-query rectangular PCPP in Theorem 2.13 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 2.1. Let δ_{Enc} be the distance of the code. Suppose a string of length n_{hard} is encoded (via Enc) into a string of length $\tilde{n}_{\mathsf{hard}} := O(n_{\mathsf{hard}})$. We set the following parameters:

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

We assume without loss of generality that $\tilde{n}_{\mathsf{hard}} = H_{\mathsf{input}} \cdot W_{\mathsf{input}}$. (This can always be done by adding at most $W_{\mathsf{input}} \le \tilde{n}_{\mathsf{hard}}$ dummy bits into the codeword of the error-correcting code, where the resulting code is still of constant rate and distance.)

We can check that the technical conditions of Theorem 2.13 for the 2-query rectangular PCPP construction holds:[15]

$$\begin{aligned}
\log T(n) &= h_{\mathsf{proof}} + w_{\mathsf{proof}} - c_{\mathsf{tm}} \log\log \ell = (c+2)\log \ell - c_{\mathsf{tm}} \log\log \ell \\
h_{\mathsf{proof}} &= \log T(n) + c_{\mathsf{tm}} \log\log \ell - w_{\mathsf{proof}} = \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}} \\
\frac{h_{\mathsf{input}}}{h_{\mathsf{proof}}} &= 1 - \frac{\Theta(\log\log T)}{\log T} \le 1 - \frac{Cm \log\log T}{\log T} \qquad \text{(if the constant in } \Theta(\cdot) \text{ is large enough)} \\
\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}} &= \frac{\lceil \log \tilde{n}_{\mathsf{hard}} \rceil - h_{\mathsf{input}}}{\log \ell} \le \frac{h_{\mathsf{proof}} + \log n + O(1) - h_{\mathsf{input}}}{\log \ell} \\
&= \frac{\log n + \Theta(h_{\mathsf{proof}} \log\log T(n))/\log T(n)}{\log \ell} \\
&= \frac{\log n}{\log \ell} + \frac{O(\log\log T(n))}{\log T(n)} \\
&\le 1 - \Omega(1).
\end{aligned}$$

By Theorem 2.13, there is a PCPP verifier VPCPP for L^{enc} with oracle access to $\Pi := \mathsf{Enc}(x) \circ \pi$, where the input $\mathsf{Enc}(x)$ is treated as a matrix of size $H_{\mathsf{input}} \times W_{\mathsf{input}}$, and the proof π is treated as a matrix of size $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$. The PCPP verifier has the following parameters:

- completeness error $= 1 - c_{\mathsf{pcp}}$,
- soundness error $= s_{\mathsf{pcp}}$,
- proximity parameter $= \delta_{\mathsf{Enc}}/3$,
- query complexity ≤ 2,

[15] Strictly speaking, we apply Theorem 2.13 with w_{proof} to obtain a PCPP with proof size $H'_{\mathsf{proof}} \times W_{\mathsf{proof}}$, where $W_{\mathsf{proof}} = 2^{w_{\mathsf{proof}}}$ and $H'_{\mathsf{proof}} = 2^{h'_{\mathsf{proof}}}$ for some $h'_{\mathsf{proof}} = \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}$. We set c_{tm} to be sufficiently large so that $h_{\mathsf{proof}} \ge h'_{\mathsf{proof}}$, hence we can assume without loss of generality that the proof size is actually $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ (recall that $H_{\mathsf{proof}} = 2^{h_{\mathsf{proof}}}$) by adding dummy bits in the proof.

- parity-check bits ≤ 2,
- total randomness $= r := \log T + O(\log\log T + m\log m)$,
- row randomness $= r_{\mathsf{row}} := h_{\mathsf{proof}} - (5/m)\log T = (c+1-\eta)\log\ell - (5c_{\mathsf{tm}}/m)\log\log\ell$,
- column randomness $= r_{\mathsf{col}} := w_{\mathsf{proof}} - (5/m)\log T = (1-\eta)\log\ell - (5c_{\mathsf{tm}}/m)\log\log\ell$,
- shared randomness $= r_{\mathsf{shared}} := (10/m)\log T + c_{\mathsf{tm}}\log\log T$
$= 2\eta\log\ell + (c_{\mathsf{tm}}/m)\log\log\ell + O(\log\log\ell + m\log m)$.

Moreover, the total number of parity-check bits and queries is at most 2, and the decision predicate $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$, which takes the parity-check bits and the answers to the queries as the input, is an OR of its input bits or their negations.

For an input $x \in L^{\mathsf{hard}} \cap \{0,1\}^{n_{\mathsf{hard}}}$, we say that x *has an easy witness* if there is a proof matrix π for the statement "$\mathsf{Enc}(x) \in L^{\mathsf{enc}}$" such that:

(completeness) $\Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed}) \text{ accepts}] \geq c_{\mathsf{pcp}}$; and

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

Description of M^{PCPP}. Now we define M^{PCPP}, which is a $(T/\log^{c_{\mathsf{hard}}} T)$-time non-deterministic algorithm that takes at most $\ell^{c+1} \leq n_{\mathsf{hard}}/10$ bits of advice. M^{PCPP} aims to reject every $x \notin L$ and accept every $x \in L$ with easy witness when appropriate advice is given.

On input $x \in \{0,1\}^{n_{\mathsf{hard}}}$, 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 p_{acc} be the acceptance probability of the PCPP verifier VPCPP for L^{enc} given the input $\mathsf{Enc}(x)$ and the proof π, i.e.,

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

We need to distinguish between the case that $p_{\mathsf{acc}} \geq c_{\mathsf{pcp}}$ and the case that $p_{\mathsf{acc}} \leq s_{\mathsf{pcp}}$. We set $\varepsilon := (c_{\mathsf{pcp}} - s_{\mathsf{pcp}})/4$ so that this can be done by estimating p_{acc} with an additive error at most ε, which will be done by applying the $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs algorithm in the assumption. (Recall that c_{pcp} and s_{pcp} are absolute constants that only depend on δ_{Enc}, which means that ε is also an absolute constant.)

In what follows, we reduce the problem of estimating p_{acc} to $2^{r_{\mathsf{shared}}}$ instances of $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs, where each instance consists of $2^{r_{\mathsf{col}}} = \ell^{1-\eta}\cdot\mathrm{polylog}(\ell)$ circuits and $2^{r_{\mathsf{row}}} = \ell^{c+1-\eta}\cdot\mathrm{polylog}(\ell)$ inputs. Then we will utilize the algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs to estimate p_{acc}, where the data structure in the preprocessing phase will be treated as an advice of M^{PCPP}.

For the simplicity of presentation, we define the notation:

$$\begin{aligned}
(\mathsf{itype}[1],\ldots,\mathsf{itype}[q]) &\leftarrow V_{\mathsf{type}}(\mathsf{seed.shared}),\\
(\mathsf{irow}[1],\ldots,\mathsf{irow}[q]) &\leftarrow V_{\mathsf{row}}(\mathsf{seed.shared},\mathsf{seed.row}),\\
(\mathsf{icol}[1],\ldots,\mathsf{icol}[q]) &\leftarrow V_{\mathsf{col}}(\mathsf{seed.shared},\mathsf{seed.col}), \qquad \text{and}\\
(pc_1,\ldots,pc_p) &\leftarrow V_{\mathsf{pc}}(\mathsf{seed.shared}),
\end{aligned}$$

where $p+q \leq 2$ and $pc_i : \{0,1\}^{r_{\mathsf{row}}+r_{\mathsf{col}}} \to \{0,1\}$ is an XOR of (some of) its input bits (i.e. a GF(2)-linear function) for every $i \in [p]$.

Reduction to Satisfying-Pairs. Our input strings in the $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs instance will be of the form $(a_1, \ldots, a_q, pc_1^{\mathsf{row}}, \ldots, pc_p^{\mathsf{row}})$. For each $j \in [q]$, the meaning of a_j is as follows:

- 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!)

For each $j \in [p]$, pc_j^{row} is a bit representing the contribution of $\mathsf{seed.row}$ in the j-th parity-check bit, i.e. $pc_j^{\mathsf{row}} := pc_j(\mathsf{seed.row}, 0^{|\mathsf{seed.col}|})$.

We first enumerate $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$. For each $\mathsf{seed.shared}$, we create an instance $\mathcal{I} := \mathcal{I}_{\mathsf{seed.shared}}$ of $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs as follows. 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 each $\mathsf{seed.row} \in \{0,1\}^{r_{\mathsf{row}}}$, we add the following input to $\mathcal{I}$:

$$\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.row}} = (a_1, \ldots, a_q, pc_1^{\mathsf{row}}, \ldots, pc_p^{\mathsf{row}}),$$

where for every $j \in [q]$,

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

and pc_j^{row} is the contribution of $\mathsf{seed.row}$ to the j-th parity-check bit as defined above. Note that since $\tilde{n}_{\mathsf{hard}} = H_{\mathsf{input}} \cdot W_{\mathsf{input}}$, $\tilde{x}_{\mathsf{irow}[j]} \in \{0,1\}^{W_{\mathsf{input}}}$ when $\mathsf{itype}[j] = \mathsf{input}$, i.e., $\tilde{x}_{\mathsf{irow}[j]}$ will not contain $\perp$ (see Definition 2.9). The length of a_j is at most $\max\{W_{\mathsf{input}}, n\} \le n \cdot \mathrm{polylog}(\ell)$, thus the total length of $\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.row}}$ is also bounded by $n \cdot \mathrm{polylog}(\ell)$.

Then, for every $\mathsf{seed.col} \in \{0,1\}^{r_{\mathsf{col}}}$, we define a circuit $C_{\mathsf{seed.shared},\mathsf{seed.col}}$ as follows. On input

$$(a_1, \ldots, a_q, pc_1^{\mathsf{row}}, \ldots, pc_p^{\mathsf{row}}),$$

it outputs

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

Here, $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ is the decision predicate of VPCPP and pc_i^{col} represents the contribution of $\mathsf{seed.col}$ to the i-th parity-check bit, i.e., $pc_i^{\mathsf{col}} := pc_i(0^{|\mathsf{seed.row}|}, \mathsf{seed.col})$. Note that by definition, $pc_i(\mathsf{seed.row}, \mathsf{seed.col}) = pc_i^{\mathsf{row}} \oplus pc_i^{\mathsf{col}}$. Also note that $C_{\mathsf{seed.shared},\mathsf{seed.col}}$ is indeed an $\mathsf{OR}_2 \circ \mathscr{C}$ circuit, since VDec is always the OR of its two input bits or their negation.

Figure 1: Examples of the circuit $C_{\mathsf{seed.shared},\mathsf{seed.col}}$. In the left example, there are two queries and no parity-check bits, the first query has type proof and the second query has type input. In the right example, there are one query with type proof and one parity-check bit.

Now, our instance $\mathcal{I}$ contains $M := 2^{r_{\mathsf{row}}}$ inputs and $N := 2^{r_{\mathsf{col}}}$ circuits. By definition, we have

$$\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed}) = C_{\mathsf{seed.shared},\mathsf{seed.col}}(\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.row}}).$$

Since $M = \ell^{c+1-\eta} \cdot \mathrm{polylog}(\ell)$ and $N = \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$, there is a non-trivial algorithm for $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs with N circuits of size s and M inputs of length $n \cdot \mathrm{polylog}(\ell)$. In particular, we can estimate $p_{\mathsf{acc}}(\mathsf{seed.shared})$ using this algorithm on $\mathcal{I}_{\mathsf{seed.shared}}$ up to an additive error ε, where

$$p_{\mathsf{acc}}(\mathsf{seed.shared}) := \Pr_{\mathsf{seed.row},\mathsf{seed.col}}\left[\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})\right].$$

In other words, we can obtain a $p'_{\mathsf{acc}}(\mathsf{seed.shared}) \in p_{\mathsf{acc}}(\mathsf{seed.shared}) \pm \varepsilon$. The overall acceptance probability of VPCPP on the input $\mathsf{Enc}(x)$ and proof π is

$$\begin{aligned}
p_{\mathsf{acc}} := \quad & \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}\left[\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})\right]. \\
= \quad & \mathop{\mathbb{E}}_{\mathsf{seed.shared}}\left[\Pr_{\mathsf{seed.row},\mathsf{seed.col}}\left[\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})\right]\right] \\
= \quad & \mathop{\mathbb{E}}_{\mathsf{seed.shared}}\left[p_{\mathsf{acc}}(\mathsf{seed.shared})\right] \\
\in \quad & \mathop{\mathbb{E}}_{\mathsf{seed.shared}}\left[p'_{\mathsf{acc}}(\mathsf{seed.shared})\right] \pm \varepsilon.
\end{aligned}$$

Hence we can estimate p_{acc} up to an additive error ε by taking average over all $p'_{\mathsf{acc}}(\mathsf{seed.shared})$ obtained by the $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs algorithm over $\mathcal{I}_{\mathsf{seed.shared}}$.

To summarise, our algorithm M^{PCPP} works as follows. It first computes $\mathsf{Enc}(x)$ in $O(n)$ time. Then, it enumerates $\mathsf{seed.shared}$, produces the instance $\mathcal{I}_{\mathsf{seed.shared}}$, and feeds it to the algorithm for $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs to obtain $p'_{\mathsf{acc}}(\mathsf{seed.shared})$. Let p'_{acc} be the average of $p'_{\mathsf{acc}}(\mathsf{seed.shared})$ over all $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$. It accepts if and only if $p'_{\mathsf{acc}} \geq c_{\mathsf{pcp}} - \varepsilon$.

Correctness of M^{PCPP}. For every $x \in \{0,1\}^{n_{\mathsf{hard}}}$, we know by the discussion above that:

- If $x \notin L^{\mathsf{hard}}$, we know that $\mathsf{Enc}(x)$ is δ_{Enc} far from being in L^{enc}. By the soundness of VPCPP, $p_{\mathsf{acc}} \leq s_{\mathsf{pcp}}$, which further means that $p'_{\mathsf{acc}} \leq p_{\mathsf{acc}} + \varepsilon < c_{\mathsf{pcp}} - \varepsilon$, hence M^{PCPP} will reject x.
- If $x \in L^{\mathsf{hard}}$ has an *easy witness*, we can see by the definition of easiness that there is a proof π of $\mathsf{Enc}(x) \in L^{\mathsf{enc}}$ such that for every row $\pi_i \in \{0,1\}^{W_{\mathsf{proof}}}$ of π, there is a string $w_i \in \{0,1\}^n$ such that $\pi_i = C(w_i)$. These w_i can be found by non-deterministic guessing at the beginning of M^{PCPP}. In such case, we know by the completeness of VPCPP that $p_{\mathsf{acc}} \geq c_{\mathsf{pcp}}$, which further means that $p'_{\mathsf{acc}} \geq p_{\mathsf{acc}} - \varepsilon \geq c_{\mathsf{pcp}} - \varepsilon$. Therefore M^{PCPP} will accept x.

Complexity of M^{PCPP}. Each instance $\mathcal{I}$ of $\mathsf{Approx}_{\varepsilon}$-$\mathscr{C}'$-Satisfying-Pairs contains $M := 2^{r_{\mathsf{row}}}$ inputs and $N := 2^{r_{\mathsf{col}}}$ circuits. Since each instance can be solved in $NM/\log^{c_{\mathsf{imp}}}(NM)$ time, the total time are

$$\begin{aligned}
& 2^{r_{\mathsf{shared}}} \cdot NM/\log^{c_{\mathsf{imp}}}(NM) \\
\leq\ & 2^{r_{\mathsf{shared}}} \cdot 2^{r_{\mathsf{row}}} \cdot 2^{r_{\mathsf{col}}}/r^{c_{\mathsf{imp}}} \\
\leq\ & 2^r/r^{c_{\mathsf{imp}}}.
\end{aligned}$$

Recall that $r = \log T + O(\log\log T + m\log m)$, where $O(\cdot)$ hides some absolute constant, we can see that $2^r/r^{c_{\mathsf{imp}}} = T\log^{O(1)} T/\log^{c_{\mathsf{imp}}} T$. By setting c_{imp} to be an sufficiently large absolute constant depending on c_{hard}, we can make $2^r/r^{c_{\mathsf{imp}}} \leq T/\log^{c_{\mathsf{hard}}} T$. Also, we can compute $\mathsf{Enc}(x)$ in

$O(n_{\mathsf{hard}})$ time and this is not the bottleneck. Therefore, the total running time of M^{PCPP} is at most $T/\log^{c_{\mathsf{hard}}} T$.

It then suffices to determine the advice and non-determinism complexity of M^{PCPP}. For every $\mathsf{seed.shared}$, the machine M^{PCPP} needs the data structure $\mathsf{DS}_{\mathsf{seed.shared}}$ as advice to support the algorithm for Satisfying Pairs. Since $|\mathsf{DS}_{\mathsf{seed.shared}}| \leq N^c = 2^{cr_{\mathsf{col}}}$ by the assumption, the advice complexity of M^{PCPP} is

$$2^{cr_{\mathsf{col}}+r_{\mathsf{shared}}} \leq \ell^{c-c\eta+2\eta} \leq \ell^{c+1} \leq n_{\mathsf{hard}}/10.$$

Also, the number of nondeterministic bits that M^{PCPP} guesses is at most $H_{\mathsf{proof}} \cdot n \leq n_{\mathsf{hard}}/10$. Therefore we can see that

$$M^{\mathsf{PCPP}} \in \mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T/\log^{c_{\mathsf{hard}}}(T), n_{\mathsf{hard}}/10]_{/(n_{\mathsf{hard}}/10)}.$$

The final algorithm. Given a multi-output circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, our algorithm for finding a non-output of C works as follows. First, we construct the hard language L^{hard} and the algorithm M^{PCPP}. Since M^{PCPP} is a nondeterministic algorithm that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, uses at most $n_{\mathsf{hard}}/10$ bits of nondeterminism and at most $n_{\mathsf{hard}}/10$ bits of advice, it follows that there is an input $x_{\mathsf{hard}} \in \{0,1\}^{n_{\mathsf{hard}}}$ such that $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) \neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$. Moreover, let α be the advice string fed to M^{PCPP}, i.e., the data structures $\mathsf{DS}_{\mathsf{seed.shared}}$ for each $\mathsf{seed.shared}$. (Note that we can obtain α since the avoidance algorithm has an NP oracle.) We can find such an input x_{hard} by running $\mathcal{R}(1^{n_{\mathsf{hard}}}, M^{\mathsf{PCPP}}, \alpha)$, where $\mathcal{R}$ is the refuter guaranteed by Theorem 2.2. Thus, we can find x_{hard} in $\mathrm{poly}(T)$ time with an NP oracle.

If $x_{\mathsf{hard}} \notin L^{\mathsf{hard}}$, then M^{PCPP} also rejects x_{hard}, which means $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) = L^{\mathsf{hard}}(x_{\mathsf{hard}})$. Thus it has to be the case that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but M^{PCPP} rejects x_{hard}. Therefore, x_{hard} does not have an easy witness. We can then use the NP oracle to find the lexicographically first proof matrix π such that

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

Treating π as a matrix of dimension $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$, there has to be a row that is not in the range of C. We can pick such a row by using the NP oracle. □

Remark 3.3. In Theorem 3.2, we assumed a non-trivial Satisfying-Pairs algorithm for the circuit class $\mathsf{OR}_2 \circ \mathscr{C}$. By Theorem 2.17, a non-trivial Satisfying-Pairs algorithm for $\mathsf{AND}_2 \circ \mathscr{C}$ or $\mathsf{XOR}_2 \circ \mathscr{C}$ also suffices. This property might be useful for some circuit classes with a better closure property under top XOR_2 gates (or AND_2 gates).

By replacing the 2-query PCPP (with imperfect completeness) with the 3-query PCPP (with perfect completeness) in Theorem 2.13, we can show that non-trivial algorithms for Gap_ε-$\mathscr{C}'$-Satisfying-Pairs where $\mathscr{C}' = \mathsf{OR}_3 \circ \mathscr{C}$ also imply $\mathsf{FP}^{\mathsf{NP}}$ algorithms for $\mathscr{C}$-Avoid. We state the result below but omit the proofs.

Corollary 3.4. *There are constants $\varepsilon > 0$ and c_{imp} such that the following holds. Let $0 < \eta < 1/2$ be a constant, $\ell(n) > n^{1+4\eta}$ be a good function. Let $s = s(n)$ be a size parameter, $\mathscr{C}[s]$ be a typical circuit class where s is a size parameter, and $\mathscr{C}'[3s] := \mathsf{OR}_3 \circ \mathscr{C}[s]$ (i.e. a $\mathscr{C}'$ circuit of size $3s$ refers to an OR_3 of at most two $\mathscr{C}$ circuits of size s).*

Assumption: *Suppose that for some constant $c \geq 1$, there is an $(NM/\log^{c_{\mathsf{imp}}}(NM))$-time algorithm for Gap_ε-$\mathscr{C}'$-Satisfying-Pairs with $N := \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$ circuits of size $s(n)$ and $M := \ell^{c+1-\eta} \cdot \mathrm{polylog}(\ell)$ inputs of length $n \cdot \mathrm{polylog}(\ell)$, allowing a P^{NP} preprocessing of an N^c-size data structure.*

Conclusion: *Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Avoid with stretch $\ell(n)$.*

3.2 Remote Point from Satisfying-Pairs

Theorem 3.5. *There is a universal constant $c_u \geq 1$ such that the following holds. Let $N := N(n)$ be a parameter such that $2^{\log^{c_u} n} < N < 2^{n^{0.99}}$, $\varepsilon := \varepsilon(n) > n^{-c_u}$ be the error parameter, and $\ell := N^{c_u \log(1/\varepsilon)}$. Let $\mathscr{C}[s]$ be a typical circuit class, where $s := s(n) \leq N$ is a size parameter, and denote $\mathscr{C}'[c_u s] := \mathsf{AND}_{c_u} \circ \mathscr{C}[s]$ (i.e. a $\mathscr{C}'$ circuit of size $c_u s$ refers to the AND of at most c_u $\mathscr{C}$ circuits of size s).*

Assumption: *Let $P := (\log N)^{\log(1/\varepsilon)}$. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} := N^2/P^{c_u}$ that, given as input a list of N $\mathscr{C}'[c_u s]$ circuits $\{C_i\}$ and a list of N inputs $\{x_j\}$ with input length $n \cdot \mathrm{polylog}(\ell)$, estimates $\Pr_{i,j \leftarrow [N]}[C_i(x_j)]$ with additive error $\eta := \varepsilon^{c_u}$.*

Conclusion: *Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where each output bit of C can be computed in $\mathscr{C}[s]$, and prints a string y that is $(1/2 - \varepsilon)$-far from $\mathsf{Range}(C)$.*

The rest of this section is devoted to proving Theorem 3.5.

Overview of Section 3.2

- In Section 3.2.1, we define a circuit class called $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$, and show that a Satisfying-Pairs algorithm for $\mathsf{AND}_d \circ \mathscr{C}$ implies a Satisfying-Pairs algorithm for this class. This will be a convenient tool for our subsequent arguments.
- To solve the remote point problem, we need to define a nondeterministic machine called M^{PCPP} trying to contradict the nondeterministic time hierarchy (Theorem 2.2). In Section 3.2.2, we set the framework for this machine: it uses the PCPP theorem in Theorem 2.14, guesses a "compressed" version of the PCPP proof, and verifies the validity of this PCPP proof without decompressing it.
- The first problem we encounter is the "non-Booleanness" of the PCPP proof. As we use Theorem 2.5, the decompressed proof consists of real numbers instead of Boolean values, and we need to check whether the decompressed proof is "close to Boolean" (in a carefully defined technical sense). This is done in Section 3.2.3 via the Satisfying-Pairs algorithm.
- In Section 3.2.4, we use the faster algorithm for Satisfying-Pairs to verify the PCPP proof. This step is straightforward but tedious.
- After we obtain a non-trivial algorithm for verifying the PCPP proof, we conclude the machine M^{PCPP} in Section 3.2.5. Then we use this machine to build an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for the remote point problem in Section 3.2.6.
- Finally, Appendix A contains postponed proofs.

3.2.1 Satisfying-Pairs for $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ Circuits

It turns out that as an intermediate step, we need a Satisfying-Pairs algorithm for the following class of multi-output circuits that output real numbers. Let $d \geq 1$ be a constant, $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ denotes the class of multi-output circuits which takes two inputs $x \in \{0,1\}^n$ and α, and has the following components:

- Let $\ell_{\mathscr{C}}$ denote the number of bottom $\mathscr{C}$ circuits. For each $i \in [\ell_{\mathscr{C}}]$, the i-th circuit is a $\mathscr{C}$ circuit computing a function $C_i : \{0,1\}^n \to \{0,1\}$.
- Let ℓ_{Sum} denote the number of middle "linear sum" gates. For each $i \in [\ell_{\mathsf{Sum}}]$, the i-th gate outputs

$$\mathsf{Sum}_i(x, \alpha) := \sum_{k \in [A]} \mathsf{coeff}_k(\alpha) \cdot C_{\mathsf{idx}_k(\alpha, i)}(x).$$

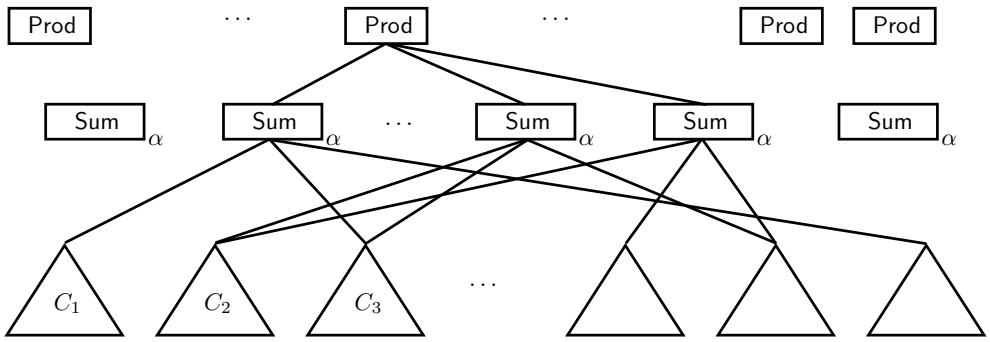

Figure 2: Example of a $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit.

(See Definition 2.3 for the definition of linear sum circuits, in particular the coefficient sum and locality of a linear sum circuit. Note that this definition is different from the definitions in [Wil18b, CW19b].)

- Let ℓ_{Prod} denote the number of output gates. Each output gate is a product (i.e., multiplication) gate of fan-in d, and is connected to the $q_1(i), q_2(i), \ldots, q_d(i)$-th linear sum gate. Its output is

$$C_i^{\mathsf{Prod}}(x, \alpha) := \prod_{t=1}^{d} \mathsf{Sum}_{q_t(i)}(x, \alpha).$$

Remark 3.6. The important measures of a $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit are:

- the number of gates in each level ($\ell_{\mathscr{C}}$, ℓ_{Sum}, ℓ_{Prod});
- the fan-in of the top Prod gates (d);
- the fan-in (A), coefficient sum (U), and locality (l) of the linear sum layer.

We show that a Satisfying-Pairs algorithm for $\mathsf{AND}_d \circ \mathscr{C}$ circuits implies a "Satisfying-Pairs algorithm" for $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuits that given a list of $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuits and a list of input strings, estimates the expected output value (as a real number) for a random circuit and a random input string in the lists. The proof is deferred to Appendix A.1.

Theorem 3.7. *Let $\mathscr{C}$ be a typical circuit class, $M' \geq 1$ and $\eta \in (0, 1)$ be parameters. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} = T^{\mathrm{alg}}(N, M)$ that, given as input a list of $\hat{N} \leq N$ $\mathsf{AND}_d \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of $\hat{M} \leq M$ inputs $\{x_j\}$, estimates the following quantity with additive error η:*

$$\Pr_{i \leftarrow [\hat{N}], j \leftarrow [\hat{M}]}[C_i(x_j)].$$

Then, there is a deterministic algorithm running in time $A^d(2^{dl} + M'/M) \cdot (\ell_{\mathsf{Prod}}/N) \cdot O(T^{\mathrm{alg}})$ that, given as input a $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit C^{Prod} with parameters specified in Remark 3.6, and a list of M' inputs $\{(x_j, \alpha_j)\}$, estimates the following quantity with additive error $\eta \cdot U^d$:

$$\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow [M']} \left[C_i^{\mathsf{Prod}}(x_j, \alpha_j) \right].$$

3.2.2 Set Up

Suppose that we are given a $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}^\ell$ as input. Let q, c_m, c_{tm} be constants that will be determined later. Define

$$\begin{aligned}
\delta &:= (10^9 q)^{-q^2}, \\
m &:= c_m \log(1/\varepsilon)/\delta, \\
w_{\mathsf{proof}} &:= (60q/m)\log \ell, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = \ell^{O(\delta/\log(1/\varepsilon))}, \\
h_{\mathsf{proof}} &:= (5q+1)\log \ell, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = \ell^{5q+1}, \\
n_{\mathsf{hard}} &:= 20 H_{\mathsf{proof}} \cdot n, \\
T &:= H_{\mathsf{proof}} \cdot W_{\mathsf{proof}} / \log^{c_{\mathsf{tm}}}(\ell).
\end{aligned}$$

Let L^{hard} be the hard language constructed in Theorem 2.2, i.e.,

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

where n_{hard} refers to the input length and c_{hard} is an absolute constant.

Since $T = \ell^{5q+1+\Theta(1/\log(1/\varepsilon))}/\log^{c_{\mathsf{tm}}}(\ell) = \ell^{5q+1} \cdot n^{\omega(1)}$ and $n_{\mathsf{hard}} = 20\ell^{5q+1}n$, we can see that $n_{\mathsf{hard}} \cdot \mathrm{polylog}(n_{\mathsf{hard}}) \le T \le 2^{\mathrm{poly}(n_{\mathsf{hard}})}$, which satisfies the technical condition of Theorem 2.2.

Like in the proof of Theorem 3.2, we describe a nondeterministic RAM M^{PCPP} that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, guesses $n_{\mathsf{hard}}/10$ nondeterministic bits, and attempts to solve L^{hard}. We will show that for every input $x \in \{0,1\}^{n_{\mathsf{hard}}}$, if $x \notin L^{\mathsf{hard}}$ then $M^{\mathsf{PCPP}}(x)$ rejects; while if $x \in L^{\mathsf{hard}}$ and *has an easy witness*, then $M^{\mathsf{PCPP}}(x)$ accepts. However, to solve $\mathscr{C}$-Remote-Point, we need a slightly different definition for "easy witness".

Let VPCPP be the verifier for the smooth and rectangular PCPP (Theorem 2.14) for the 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 2.1. Let δ_{Enc} be the (relative) distance of the error-correcting code. Suppose a string of length n_{hard} is encoded (via Enc) into a string of length $\tilde{n}_{\mathsf{hard}} := \Theta(n_{\mathsf{hard}})$. We set the following parameters:

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

Again, we assume without loss of generality that $\tilde{n}_{\mathsf{hard}} = H_{\mathsf{input}} \cdot W_{\mathsf{input}}$.

We invoke Theorem 2.14 for L^{enc} to obtain a verifier VPCPP with proof size $\hat{H}_{\mathsf{proof}} \times W_{\mathsf{proof}}$ and input size $H_{\mathsf{input}} \times W_{\mathsf{input}}$, where $\hat{H}_{\mathsf{proof}} = 2^{\hat{h}_{\mathsf{proof}}}$ for some[16] $\hat{h}_{\mathsf{proof}} = \log T + \Theta(m \log\log T) - w_{\mathsf{proof}}$. We can check the technical requirements of Theorem 2.14 as follows:

$$\begin{aligned}
T &\ge H_{\mathsf{proof}} \cdot \ell^{\Theta(\delta/\log(1/\varepsilon))} \ge H_{\mathsf{proof}} \cdot \Theta(n) = \tilde{n}_{\mathsf{hard}}, \\
m &= \Theta(\log n/\delta) \le (\log T)^{0.1}, \\
w_{\mathsf{proof}} &= (60q/m)\log \ell \ge (5/m)\log T, \\
\hat{h}_{\mathsf{proof}} &= h_{\mathsf{proof}} + \Theta(\log n \log\log \ell) \ge (5q+1)\log \ell \ge (5/m)\log T,
\end{aligned}$$

[16] Note that the function $\hat{h}_{\mathsf{proof}}$ that Theorem 2.14 produces might not be exactly equal to h_{proof}. However, this difference is minor as these two quantities are close to each other.

$$\begin{aligned}\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}} &= \frac{h_{\mathsf{proof}} + \log n + O(1) - h_{\mathsf{input}}}{w_{\mathsf{proof}}} \\ &= \frac{\log n}{w_{\mathsf{proof}}} + \frac{h_{\mathsf{proof}}}{w_{\mathsf{proof}}} \cdot \frac{\Theta(m^2 \log\log T)}{\log T} \\ &\le \frac{m \log n}{\log \ell} + \frac{\Theta(m^3 \log\log T)}{\log T} \\ &\le 1 - \Omega(1).\end{aligned}$$

By Theorem 2.14, VPCPP has the following parameters:

- soundness error $= 1/2$,
- proximity parameter $= \delta_{\mathsf{Enc}}$,
- query complexity $= q := O(1)$,
- parity-check complexity $= q := O(1)$,
- total randomness $= r := \log T + O(\log\log T + m \log m)$,
- row randomness $= r_{\mathsf{row}} := \hat{h}_{\mathsf{proof}} - (5/m)\log T = \Theta(\log \ell)$,
- column randomness $= r_{\mathsf{col}} := w_{\mathsf{proof}} - (5/m)\log T = \Theta(\log \ell/m)$,
- shared randomness $= r_{\mathsf{shared}} := (10/m)\log T + O(\log\log T + m\log m) = \Theta(\log \ell/m)$.

Here, all the $\Theta(\cdot)$ hides constants that may depend on q, c_m, c_{tm}. Moreover, as we choose the soundness error and the proximity parameters to be absolute constants, the query complexity q is also an absolute constant.

Note that if c_u is large enough, then we have $2^{\Omega(r_{\mathsf{col}})} \le N \le 2^{r_{\mathsf{col}}}$. Therefore we can solve the Approx_η-Satisfying-Pairs problem for $2^{r_{\mathsf{col}}}$ inputs and $2^{r_{\mathsf{col}}}$ $\mathsf{AND}_{c_u} \circ \mathscr{C}$ circuits, by partitioning the inputs and circuits into groups of size N. The time complexity is still at most $2^{2r_{\mathsf{col}}}/P^{c_u}$, where $P := (r_{\mathsf{col}})^{\log(1/\varepsilon)}$. Without loss of generality, we may assume $N = 2^{r_{\mathsf{col}}}$ in what follows.

We also fix the hardness amplification procedure $\mathsf{Amp} : \{0,1\}^{W_{\mathsf{proof}}} \to \{0,1\}^{\ell'}$ described by Theorem 2.5 that amplifies hardness δ to hardness $(1/2 - \varepsilon)$. Here, $\ell' := W_{\mathsf{proof}}^{O(\log(1/\varepsilon)/\delta)} = \ell^{O(60q/c_m)}$. We set the parameter c_m such that $\ell' \le \ell$. Without loss of generality, we may assume that $\ell' = \ell$.[17] Let $(\mathsf{idx}, \mathsf{coeff})$ be the family of linear sum circuit described in Theorem 2.5, then $(\mathsf{idx}, \mathsf{coeff})$ has the following parameters:

$$\begin{aligned}\text{advice complexity} &= a := O(\log^2 W_{\mathsf{proof}}/(\varepsilon\delta)^2) = O(\log^2 \ell/\varepsilon^2), \\ \text{fan-in} &= A := O(\log W_{\mathsf{proof}}/(\varepsilon\delta)^2) = O(\log \ell/\varepsilon^2), \\ \text{coefficient sum} &= U := O(1/\varepsilon), \\ \text{locality} &= l := \log \ell.\end{aligned}$$

We say an input x *has an easy witness* if there is a proof matrix π such that:

(completeness) for every $\mathsf{seed} \in \{0,1\}^r$, $\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})$ accepts;

(*approximate* easiness) for every row π_i of π, there exists an input $w_i \in \{0,1\}^n$ and an advice $\alpha_i \in \{0,1\}^a$ such that the decoding of $C(w_i)$ with advice α_i is δ-close to π_i with respect to ℓ_1-norm. (Recall that $\mathsf{dec}_\alpha(x)$ denotes the decoding of x under advice α.) In particular:

1. for every $j \in [W_{\mathsf{proof}}]$, $(\mathsf{dec}_{\alpha_i}(C(w_i)))_j \in [0,1]$;

[17] If $\ell' \ll \ell$, we can partition the outputs of the circuit into blocks of size ℓ', and solve the remote point problem for each block of output bits.

2. $\|\mathsf{dec}_{\alpha_i}(C(w_i)) - \pi_i\|_1 \le \delta$.

Recall that $P = (r_{\mathsf{col}})^{\log(1/\varepsilon)}$. By our hypothesis, there is an algorithm that takes as input a list of N $\mathsf{AND}_{4q} \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of N inputs $\{x_j\}$, runs in deterministic $T^{\mathrm{alg}} := 2^{2r_{\mathsf{col}}}/P^{c_u}$ time, and estimates $\mathbb{E}_{i,j}[C_i(x_j)]$ within additive error $\eta := \varepsilon^{c_u} \le U^{-10q}$.

3.2.3 Guessing and Verifying the PCPP

On input $x \in \{0,1\}^{n_{\mathsf{hard}}}$, we guess $\hat{H}_{\mathsf{proof}}$ strings $w_1, w_2, \ldots, w_{\hat{H}_{\mathsf{proof}}} \in \{0,1\}^n$ as well as $\hat{H}_{\mathsf{proof}}$ advice strings $\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}} \in \{0,1\}^a$. Let $\pi_i^{\mathsf{Real}} := \mathsf{dec}_{\alpha_i}(C(w_i))$, and π_i^{Bool} be the Boolean string that is closest to π_i^{Real}. We will think of the matrix π^{Bool} as the PCPP proof, although our algorithm M^{PCPP} will operate on π^{Real}.

Therefore, before we proceed, we need to verify that π^{Real} and π^{Bool} are "close", so that it does no harm to operate on π^{Real} even if the correct PCPP proof should be π^{Bool}. This verification phase also occurs in previous works proving lower bounds against linear combinations of circuits [Wil18b, CW19b, CR22, CLW20]. Like in previous work, we only provide an "approximate" verification algorithm: if the input has an easy witness, then the PCPP proof π^{Real} corresponding to this easy witness is accepted; on the other hand, we reject every π^{Real} that is "too far" from Boolean.

In what follows, denote

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

For each $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ and each $\iota \in [q]$ such that $\mathsf{itype}[\iota] = \mathsf{proof}$, we define the following functions:

$$\begin{aligned}f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota}(\mathsf{seed.row}, \mathsf{seed.col}) &= \pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} \quad \text{and}\\ f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}(\mathsf{seed.row}, \mathsf{seed.col}) &= \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}.\end{aligned}$$

We will speak about the ℓ_d-norms of the above functions. For example, let $d \in \mathbb{N}$ be a constant, then

$$\|f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota} - f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}} \left[\left|\pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} - \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}\right|^d\right]^{1/d}.$$

Lemma 3.8. *Let $\mathscr{C}$ be a typical circuit class and $d \ge 2$ be an even number. Suppose there is an algorithm that takes as input a list of $N = 2^{r_{\mathsf{col}}}$ $\mathsf{AND}_{2d} \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of N inputs $\{x_j\}$, runs in deterministic T^{alg} time, and estimates the following quantity with additive error η:*

$$\Pr_{i,j\leftarrow[2^{r_{\mathsf{col}}}]}[C_i(x_j)].$$

Then there is an algorithm that takes the circuit C, $(w_1, w_2, \ldots, w_{\hat{H}_{\mathsf{proof}}})$, and $(\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}})$ as input, runs in deterministic $O((3A)^{2d}T^{\mathrm{alg}}) \cdot \left(2^{2dl+r_{\mathsf{shared}}} + T\log^{O(m)} T/2^{2r_{\mathsf{col}}}\right)$ time, and satisfies the following:

(Completeness) *If for every $i \in [\hat{H}_{\mathsf{proof}}]$, it holds that (1) for every $j \in [W_{\mathsf{proof}}]$, $\pi^{\mathsf{Real}}_{i,j} \in [0,1]$; (2) $\|\pi_i^{\mathsf{Real}} - \pi_i^{\mathsf{Bool}}\|_1 \le \delta$, then the algorithm accepts.*

(Soundness) *If the algorithm accepts, then it holds that*

1. *for every* $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ *and* $\iota \in [q]$, $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d \le 1 + 2\eta \cdot U^d$;
2. $\mathbb{E}_{i \leftarrow [\hat{H}_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]}\left[|\pi^{\mathsf{Real}}_{i,j} - \pi^{\mathsf{Bool}}_{i,j}|^d\right] \le 4^d \cdot \delta + 2^{d+1}\eta(2U+1)^{2d}$.

To prove this lemma, we need to reduce the task of checking whether the real proof is close to the Boolean proof to the satisfying pairs of $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuits, and then apply the non-trivial algorithm in Theorem 3.7. The details are given in Appendix A.2.

We substitute $d := 2q$ in the above lemma. If x has an easy witness, then there is some $(w_1, w_2, \ldots, w_{\hat{H}_{\mathsf{proof}}})$ and $(\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}})$ that passes the test; on the other hand, if the test is passed, then both soundness properties in Lemma 3.8 hold:

1. for every $\mathsf{seed.shared}$ and ι, $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_{2q}^{2q} \le 1 + 2\eta \cdot U^{2q}$;
2. $\mathbb{E}_{i \leftarrow [H_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]}\left[|\pi^{\mathsf{Real}}_{i,j} - \pi^{\mathsf{Bool}}_{i,j}|^{2q}\right] \le 16^q \cdot \delta + 128^q \eta U^{4q}$.

3.2.4 Estimating the Acceptance Probability

After checking that the PCPP proof is "close to Boolean", the next step is to use it to speed up L^{hard}. We estimate

$$p_{\mathsf{acc}} := \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}\left[\mathsf{VPCPP}^{\mathsf{Enc}(x) \circ \pi^{\mathsf{Bool}}}(\mathsf{seed}) \text{ accepts}\right].$$

(Indeed, it suffices to distinguish between the case that $p_{\mathsf{acc}} \ge 5/6$ and the case that $p_{\mathsf{acc}} < 1/2$ as we will explain later.)

We enumerate $\mathsf{seed.shared}$. After fixing $\mathsf{seed.shared}$, each $\mathsf{itype}[\iota]$ is completely fixed, each $\mathsf{irow}[\iota]$ only depends on $\mathsf{seed.row}$, and each $\mathsf{icol}[\iota]$ only depends on $\mathsf{seed.col}$. We now need to estimate

$$p_{\mathsf{acc}}(\mathsf{seed.shared}) := \Pr_{\substack{\mathsf{seed.row} \leftarrow \{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col} \leftarrow \{0,1\}^{r_{\mathsf{col}}}}}\left[\mathsf{VPCPP}^{\mathsf{Enc}(x) \circ \pi^{\mathsf{Bool}}}(\mathsf{seed}) \text{ accepts}\right].$$

Suppose that we also fix $\mathsf{seed.row}$. Then, we know the q rows of the input matrix $\mathsf{Enc}(x)$ and the proof matrix π that could influence the PCPP verifier. We call them $\mathsf{row}^{\mathsf{Bool}}_1, \mathsf{row}^{\mathsf{Bool}}_2, \ldots, \mathsf{row}^{\mathsf{Bool}}_q$. In particular, for each $\iota \in [q]$:

$$\mathsf{row}^{\mathsf{Bool}}_\iota = \begin{cases} \tilde{x}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{input}, \\ \pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{proof}. \end{cases}$$

We also let $pc_1, pc_2, \ldots, pc_q \leftarrow V_{\mathsf{pc}}(\mathsf{seed.shared})$ be the parity-check functions of the PCPP verifier, where each $pc_\iota : \{0,1\}^{r_{\mathsf{row}}+r_{\mathsf{col}}} \to \{0,1\}$. In particular, let pc^{row}_ι (resp. pc^{col}_ι) denote the contribution of $\mathsf{seed.row}$ (resp. $\mathsf{seed.col}$) to pc_ι, i.e.,

$$pc^{\mathsf{row}}_\iota(\mathsf{seed.row}) := pc_\iota(\mathsf{seed.row}, 0^{r_{\mathsf{col}}}),$$
$$pc^{\mathsf{col}}_\iota(\mathsf{seed.col}) := pc_\iota(0^{r_{\mathsf{row}}}, \mathsf{seed.col}).$$

Then $pc_\iota(\mathsf{seed.row}, \mathsf{seed.col}) = pc^{\mathsf{row}}_\iota(\mathsf{seed.row}) \oplus pc^{\mathsf{col}}_\iota(\mathsf{seed.col})$. For simplicity, we omit $\mathsf{seed.row}$ and $\mathsf{seed.col}$ when they are clear from the context.

Let VDec be the decision predicate of the PCPP verifier; note that as $\mathsf{seed.shared}$ is fixed, $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ is also fixed. The input of VDec includes the answers to the q queries and the parity-check bits $pc_1, \ldots, pc_q$. On $\mathsf{seed.row}$ and $\mathsf{seed.col}$, the PCPP verifier outputs

$$\mathsf{VDec}\Big((\mathsf{row}^{\mathsf{Bool}}_1)_{\mathsf{icol}[1]}, (\mathsf{row}^{\mathsf{Bool}}_2)_{\mathsf{icol}[2]}, \ldots, (\mathsf{row}^{\mathsf{Bool}}_q)_{\mathsf{icol}[q]}, pc_1, pc_2, \ldots, pc_q\Big).$$

As every Boolean function over $2q$ bits can be written as a degree-$2q$ polynomial over the reals, we write

$$\mathsf{VDec}(a_1, a_2, \ldots, a_q, pc_1, pc_2, \ldots, pc_q) = \sum_{S \subseteq [q], S' \subseteq [q]} \theta_{S,S'} \left(\prod_{\iota \in S} a_\iota \right) \cdot \left(\prod_{\iota \in S'} pc_\iota \right),$$

where $\theta_{S,S'} \in [-2^{2q}, 2^{2q}]$. Now, define

$$p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') := \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row} \leftarrow \{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col} \leftarrow \{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{\iota \in S} (\mathsf{row}_\iota^{\mathsf{Bool}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} pc_\iota \right].$$

We have

$$p_{\mathsf{acc}}(\mathsf{seed.shared}) = \sum_{S \subseteq [q], S' \subseteq [q]} \theta_{S,S'} p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S'),$$

thus it suffices to estimate each $p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$.

Fix S and S'. Since we only have access to a real proof matrix π^{Real} instead of a Boolean proof matrix, we use the following number as an estimation of $p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$, with the only difference being π_i^{Bool} being replaced by π^{Real}:

$$p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S') = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row} \leftarrow \{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col} \leftarrow \{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{\iota \in S} (\mathsf{row}_\iota^{\mathsf{Real}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} pc_\iota \right],$$

where

$$\mathsf{row}_\iota^{\mathsf{Real}} = \begin{cases} \tilde{x}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{input}, \\ \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{proof}. \end{cases}$$

The following claim bounds the accuracy of the estimation given the ℓ_d-distance between the functions $f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota}$ and $f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}$. The proof is deferred to Appendix A.3.

Claim 3.9. *For every $S, S' \subseteq [q]$,*

$$|p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') - p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S')| \leq (1 + \delta_{\mathsf{seed.shared}})^{2q-1} \cdot \delta_{\mathsf{seed.shared}}.$$

Here,

$$\begin{aligned} \delta_{\mathsf{seed.shared}} &:= \sum_{\iota : \mathsf{itype}[\iota] = \mathsf{proof}} \| f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota} - f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota} \|_{2q} \\ (\text{recall}) &= \sum_{\iota : \mathsf{itype}[\iota] = \mathsf{proof}} \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row} \leftarrow \{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col} \leftarrow \{0,1\}^{r_{\mathsf{col}}}}} \left[\left| \pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} - \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} \right|^{2q} \right]^{1/(2q)}. \end{aligned}$$

Now we fix S, S' and estimate $p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S')$. Let $d_S := |S|, d_{S'} := |S'|$, it is without loss of generality to assume that $S = \{1, 2, \ldots, d_S\}$ and $S' = \{1, 2, \ldots, d_{S'}\}$. We construct a $\mathsf{Prod}_{d_S + d_{S'}} \circ \mathsf{Sum} \circ \mathscr{C}$ circuit $C^{\mathsf{Prod}} := C^{\mathsf{Prod}}_{\mathsf{seed.shared},S,S'}$, as well as a list of inputs $(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}})$, such that

$$C^{\mathsf{Prod}}_{\mathsf{seed.col}}(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}}) = \prod_{\iota \in S} (\mathsf{row}_\iota^{\mathsf{Real}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} pc_\iota. \tag{5}$$

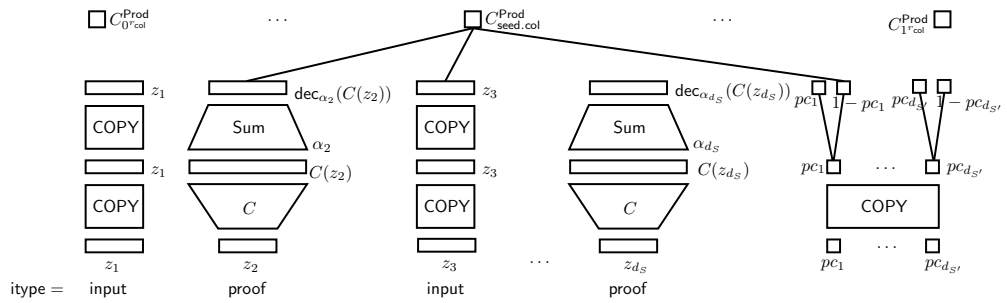

Figure 3: Construction of the circuit C^{Prod}. Note that for convenience, we only drew the "relevant" parts of this circuit, e.g., $C(z_i)$ when $\mathsf{itype}[i] = \mathsf{proof}$ and the copying circuit for z_i when $\mathsf{itype}[i] = \mathsf{input}$.

Construction of C^{Prod} and inputs. For each $\mathsf{seed.row}$ and each $\iota \in S$, define

$$(z_{\mathsf{seed.row}})_\iota := \begin{cases} \tilde{x}_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{input}, \\ w_{\mathsf{irow}[\iota]} & \text{if } \mathsf{itype}[\iota] = \mathsf{proof}. \end{cases}$$

Then we concatenate each $(z_{\mathsf{seed.row}})_j$ and the (row-)parity-check bits to obtain

$$z_{\mathsf{seed.row}} := \Big((z_{\mathsf{seed.row}})_1, (z_{\mathsf{seed.row}})_2, \ldots, (z_{\mathsf{seed.row}})_{d_S}, pc_1^{\mathsf{row}}, pc_2^{\mathsf{row}}, \ldots, pc_{d_{S'}}^{\mathsf{row}}\Big).$$

It is easy to check that given $\mathsf{seed.row}$ (and $\mathsf{seed.shared}$), we can compute $z_{\mathsf{seed.row}}$ easily. We also define

$$\alpha_{\mathsf{seed.row}} := \big(\alpha_{\mathsf{irow}[1]}, \alpha_{\mathsf{irow}[2]}, \ldots, \alpha_{\mathsf{irow}[d_S]}\big).$$

Next, we define the circuit C^{Prod} that takes two inputs (z, α) and outputs $2^{|\mathsf{seed.col}|}$ real numbers. Fix $\mathsf{seed.col}$, we want that

$$\begin{aligned} & C^{\mathsf{Prod}}_{\mathsf{seed.col}}(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}}) \\ &= \prod_{\iota \in S} (\mathsf{row}_\iota^{\mathsf{Real}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} pc_\iota \\ &= \prod_{\iota \in S^{\mathsf{proof}}} (\mathsf{dec}_{\alpha_{\mathsf{irow}[\iota]}}(C(w_{\mathsf{irow}[\iota]})))_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S^{\mathsf{input}}} \tilde{x}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} (pc_\iota^{\mathsf{row}} \oplus pc_\iota^{\mathsf{col}}), \end{aligned}$$

where

$$(\mathsf{dec}_{\alpha_{\mathsf{irow}[\iota]}}(C(w_{\mathsf{irow}[\iota]})))_{\mathsf{icol}[\iota]} = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_{\mathsf{irow}[\iota]}) \cdot C_{\mathsf{idx}_k(\alpha_{\mathsf{irow}[\iota]}, \mathsf{icol}[\iota])}(w_{\mathsf{irow}[\iota]})$$

denotes the $\mathsf{icol}[\iota]$-th bit of the string obtained by decoding $C(w_{\mathsf{irow}[\iota]})$ with the advice $\alpha_{\mathsf{irow}[\iota]}$ using the decoder dec, $S^{\mathsf{proof}} := S \cap \{i \in [q] \mid \mathsf{itype}[i] = \mathsf{proof}\}$, $S^{\mathsf{input}} := S \cap \{i \in [q] \mid \mathsf{itype}[i] = \mathsf{input}\}$.

This motivates the definition of the circuit C^{Prod} (see Figure 3 for graphic exposition and Figure 4 for detailed definition). The parameters of the circuit C^{Prod} are as follows.

- The number of gates: $\ell_{\mathscr{C}} = d_S(\ell + W_{\mathsf{proof}}) + d_{S'}$, $\ell_{\mathsf{Sum}} = W_{\mathsf{proof}} \cdot d_S + 2d_{S'}$, $\ell_{\mathsf{Prod}} = 2^{r_{\mathsf{col}}}$.
- The fan-in of the top Prod gates $d_S + d_{S'} \le 2q$.
- The fan-in $A' := A \cdot d_S + 2$, coefficient sum $d_S \cdot U + 2$, and locality l of the linear sum layer.

Circuit C^{Prod}

(Inputs) The input z will have the form $z = (z_1, z_2, \ldots, z_{d_S}, pc_1, pc_2, \ldots, pc_{d_{S'}})$ and the input α will have the form $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_{d_S})$. The intended meanings are $z_i = (z_{\mathsf{seed.row}})_i$, $pc_i = pc_i^{\mathsf{row}}$, and $\alpha_i = \alpha_{\mathsf{irow}[i]}$.

(Bottom circuits) We make d_S copies to C, where the i-th copy is applied to the input z_i. (The i-th copy is useful only when $\mathsf{itype}[i] = \mathsf{proof}$, but we make all d_S copies for convenience.) We also add $W_{\mathsf{proof}} \cdot d_S + d_{S'}$ gates to copy the input.

Thus, there are $\ell_{\mathscr{C}} := d_S \cdot \ell + d_S \cdot W_{\mathsf{proof}} + d_{S'}$ output gates; we identify $[\ell_{\mathscr{C}}]$ with the disjoint union of $\{1\} \times [d_S] \times [\ell]$, $\{2\} \times [d_S] \times [W_{\mathsf{proof}}]$, and $\{3\} \times [d_{S'}]$.

- For each $j \in [d_S]$ and $i \in [\ell]$, the $(1,j,i)$-th gate is $C_{(1,j,i)}(z) := C(z_j)_i$.
- For each $j \in [d_S]$ and $i \in [W_{\mathsf{proof}}]$, the $(2,j,i)$-th gate is $C_{(2,j,i)}(z) := (z_j)_i$.
- For each $j \in [d_{S'}]$, the $(3,j)$-th gate is $C_{3,j}(z) := pc_j$.

(Intermediate linear sum gates) There are $\ell_{\mathsf{Sum}} := W_{\mathsf{proof}} \cdot d_S + 2d_{S'}$ linear sum gates and we identify $[\ell_{\mathsf{Sum}}]$ with the disjoint union of $[W_{\mathsf{proof}}] \times [d_S]$ and $[d_{S'}] \times \{0,1\}$.

Let $i \in [W_{\mathsf{proof}}]$ and $j \in [d_S]$. If $\mathsf{itype}[j] = \mathsf{proof}$, then the (i,j)-th intermediate gate is

$$\mathsf{Sum}_{(i,j)}(z, \alpha) = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_j) \cdot C_{\mathsf{idx}_k(\alpha_j, i) \cdot d_S + j}(z).$$

It is easy to verify that

$$\mathsf{Sum}_{(i,j)}(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}}) = (\mathsf{dec}_{\alpha_{\mathsf{irow}[j]}}(C(w_{\mathsf{irow}[j]})))_i.$$

On the other hand, if $\mathsf{itype}[j] = \mathsf{input}$, then the (i,j)-th intermediate gate is $\mathsf{Sum}_{(i,j)}(z, \alpha) := (z_j)_i$. (If $i > W_{\mathsf{input}}$ then we simply set $\mathsf{Sum}_{(i,j)}(z, \alpha) = 0$ and this intermediate gate would not be used.) Finally, for each $i \in [d_{S'}]$, we have two intermediate gates

$$\mathsf{Sum}_{(i,0)}(z, \alpha) = pc_i, \quad \mathsf{Sum}_{(i,1)}(z, \alpha) = 1 - pc_i.$$

Implementation of the linear sum layer: The linear sum has fan-in $A' := A \cdot d_S + 2$ and we identify $[A']$ with the disjoint union of $[A] \times [d_S]$ and $\{+, -\}$. Let idx' and coeff' be the idx and coeff functions of the linear sum layer of C^{Prod}, then

(Function $\mathsf{idx}'_k(\alpha, i)$) Suppose $i = (i', j) \in [W_{\mathsf{proof}}] \times [d_S]$. If $\mathsf{itype}[j] = \mathsf{proof}$ and $k = (k', j')$ where $j = j'$, then we return $\mathsf{idx}'_k(\alpha, i) = (1, j, \mathsf{idx}_{k'}(\alpha_j, i'))$; if $\mathsf{itype}[j] = \mathsf{input}$ and $k = +$, then $\mathsf{idx}'_k(\alpha, i) = (2, j, i')$. Otherwise $\mathsf{idx}'_k(\alpha, i) = \mathsf{ZERO}$.

On the other hand, suppose $i = (j, b) \in [d_{S'}] \times \{0,1\}$. If ($b = 0$ and $k = +$) or ($b = 1$ and $k = -$) then $\mathsf{idx}'_k(\alpha, i) = (3, j)$. If $b = 1$ and $k = +$ then $\mathsf{idx}'_k(\alpha, i) = \mathsf{ONE}$. Otherwise $\mathsf{idx}'_k(\alpha, i) = \mathsf{ZERO}$.

(Function $\mathsf{coeff}'_k(\alpha)$) If $k = +$ then $\mathsf{coeff}'_k(\alpha) = 1$; if $k = -$ then $\mathsf{coeff}'_k(\alpha) = -1$; otherwise, if $k = (k', j')$ then $\mathsf{coeff}'_k(\alpha) = \mathsf{coeff}_{k'}(\alpha_{j'})$.

The locality of $(\mathsf{idx}', \mathsf{coeff}')$ is still l. The coefficient sum becomes $d_S \cdot U + 2$.

(Output product gates) There are $2^{r_{\mathsf{col}}}$ product gates. For each $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output gate is

$$C^{\mathsf{Prod}}_{\mathsf{seed.col}}(z, \alpha) = \prod_{i \in S} \mathsf{Sum}_{\mathsf{icol}[i] \cdot d_S + i}(z, \alpha) \cdot \prod_{i \in S'} \mathsf{Sum}_{W_{\mathsf{proof}} \cdot d_S + 2i + pc_i^{\mathsf{col}}}(z, \alpha).$$

Figure 4: Detailed definition of C^{Prod}.

Given the above construction, it is easy to check that Eq. (5) holds for every $\mathsf{seed.row}$ and $\mathsf{seed.col}$. We can see that

$$p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}}\left[C'^{\mathsf{Prod}}_{\mathsf{seed.col}}(z_{\mathsf{seed.row}}, \alpha_{\mathsf{seed.row}})\right].$$

Since $d_S \le q$ and $d_{S'} \le q$, by Theorem 3.7, we can estimate $p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$ with additive error $\eta \cdot (qU+2)^{2q}$ in deterministic $O((qA)^{2q}(2^{2ql} + 2^{r_{\mathsf{row}}}/N) \cdot T^{\mathrm{alg}})$ time.

Analysis. First, the verification step takes

$$\begin{aligned}
&O((3A)^{4q}T^{\mathrm{alg}}) \cdot \left(2^{4ql+r_{\mathsf{shared}}} + T\log^{O(m)} T/2^{2r_{\mathsf{col}}}\right)\\
\le\ &O((3A)^{4q}) \cdot (T\log^{O(m)} T)/(r_{\mathsf{col}})^{c_u \log(1/\varepsilon)}\\
\le\ &T(\log T)^{O(m)-c_u\log(1/\varepsilon)/2}
\end{aligned}$$

time, which is at most $T/\log^{c_{\mathsf{hard}}} T$ if c_u is a large enough constant.

Our algorithm estimates $p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$. By Claim 3.9, the same algorithm estimates $p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')$ within additive error $\eta(qU+2)^{2q} + \delta'_{\mathsf{seed.shared}}$, where

$$\delta'_{\mathsf{seed.shared}} := (1+\delta_{\mathsf{seed.shared}})^{2q-1} \cdot \delta_{\mathsf{seed.shared}}.$$

Running this algorithm for every possible (S, S'), we obtain an algorithm that runs in deterministic $(O(qA))^{2q}(2^{2ql} + 2^{r_{\mathsf{row}}}/M) \cdot T^{\mathrm{alg}}$ time and estimates $p_{\mathsf{acc}}(\mathsf{seed.shared})$ within additive error

$$\begin{aligned}
\le \bar{\delta}_{\mathsf{seed.shared}} &:= 4^q \cdot \sum_{S,S'}(\eta \cdot (qU+2)^{2q} + \delta'_{\mathsf{seed.shared}})\\
&\le \eta \cdot (4qU+8)^{2q} + 16^q \cdot \delta'_{\mathsf{seed.shared}}.
\end{aligned}$$

Finally, running this algorithm for every $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$, we obtain an algorithm that runs in deterministic

$$\begin{aligned}
&O(qA)^{2q}(2^{2ql} + 2^{r_{\mathsf{row}}}/N) \cdot 2^{r_{\mathsf{shared}}} \cdot O(T^{\mathrm{alg}})\\
\le\ &O(\log^{2q}\ell/\varepsilon^{4q})2^{r_{\mathsf{row}}}/2^{r_{\mathsf{col}}} \cdot 2^{r_{\mathsf{shared}}} \cdot 2^{2r_{\mathsf{col}}}/(r_{\mathsf{col}})^{c_u\log(1/\varepsilon)}\\
\le\ &2^r/\log^{\Omega(c_u)}\ell < T/\log^{c_{\mathsf{hard}}}(T)
\end{aligned}$$

time that estimates p_{acc} within additive error of at most

$$\mathop{\mathbb{E}}_{\mathsf{seed.shared}}\left[\bar{\delta}_{\mathsf{seed.shared}}\right] \le \eta \cdot (4qU+8)^{2q} + 16^q \mathop{\mathbb{E}}_{\mathsf{seed.shared}}[\delta'_{\mathsf{seed.shared}}].$$

It remains to upper bound the quantity $\mathbb{E}_{\mathsf{seed.shared}}[\delta'_{\mathsf{seed.shared}}]$. We abstract this task in the following lemma and defer the proof to Appendix A.4.

Lemma 3.10. *Let $f : [N] \times [q] \to \mathbb{R}_{\ge 0}$ be a function and $d \ge 1$ be a constant. Suppose that*

1. *for every $s \in [N]$ and $i \in [q]$, $f(s,i) \le \alpha$ (where $\alpha \ge 1$);*
2. *$\mathbb{E}_{s,i}[f(s,i)^d] \le \delta$.*

Let $f(s) := \sum_{i\in[q]} f(s,i)$. Then

$$\mathop{\mathbb{E}}_{s}[(1+f(s))^{d-1} \cdot f(s)] \le q\delta^{1/d}(2q\alpha)^{d-1}.$$

To see how this lemma corresponds to our scenario: Let $d = 2q$ and $[N] = \{0,1\}^{r_{\mathsf{shared}}}$. For $\mathsf{seed.shared}$ and ι, if $\mathsf{itype}[\iota] = \mathsf{proof}$, then define $f(\mathsf{seed.shared}, \iota) = \|f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota} - f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_{2q}$; otherwise define $f(\mathsf{seed.shared}, \iota) = 0$. Since the verification algorithm did not reject π^{Real}, we have

1. For every $\mathsf{seed.shared}$ and ι,
$$f(\mathsf{seed.shared}, \iota) \leq \|f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota}\|_{2q} + \|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_{2q} \leq 2 + 2\eta \cdot U^{2q}.$$
2. Since the PCPP is smooth, the distribution of $(\mathsf{irow}[\iota], \mathsf{icol}[\iota])$ for random (seed, ι) (conditioned on $\mathsf{itype}[\iota] = \mathsf{proof}$) is the same as the uniform distribution over $[H_{\mathsf{proof}}] \times [W_{\mathsf{proof}}]$. Therefore
$$\begin{aligned}\mathop{\mathbb{E}}_{\mathsf{seed.shared},\iota}[f(\mathsf{seed.shared}, \iota)^{2q}] &= \mathop{\mathbb{E}}_{\mathsf{seed},\iota:\mathsf{itype}[\iota]=\mathsf{proof}}[|\pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} - \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}|^{2q}] \\ &= \mathop{\mathbb{E}}_{i,j}[|\pi^{\mathsf{Bool}}_{i,j} - \pi^{\mathsf{Real}}_{i,j}|^{2q}] \\ &\leq 16^q \cdot \delta + 128^q \eta U^{4q}.\end{aligned}$$

It follows from Lemma 3.10 that

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathsf{seed.shared}}[\delta'_{\mathsf{seed.shared}}] &\leq q(16^q \cdot \delta + 128^q \eta U^{4q})^{1/2q}(2q(2 + 2\eta \cdot U^{2q}))^{2q-1} \\ &\leq q(17^q \cdot \delta)^{1/2q} \cdot (100q)^{2q} < 100^{-q}.\end{aligned}$$

Therefore, the algorithm estimates p_{acc} within additive error at most

$$\eta(4qU + 8)^{2q} + 16^q \cdot 100^{-q} < 1/6,$$

thus successfully distinguishes between the case that $p_{\mathsf{acc}} > 5/6$ and that $p_{\mathsf{acc}} < 1/2$.

3.2.5 Wrap Up: Description of M^{PCPP}

On input x, we consider the smooth and rectangular PCPP for the language $L^{\mathsf{enc}} = \{\mathsf{Enc}(x) : x \in L^{\mathsf{hard}}\}$. (Recall that M^{PCPP} aims to reject every $x \notin L$ and accepts every $x \in L$ with easy witness.) We guess $(w_1, \ldots, w_{H_{\mathsf{proof}}})$ and $(\alpha_1, \ldots, \alpha_{H_{\mathsf{proof}}})$, which implicitly defines the PCPP proof matrices π^{Bool} and π^{Real}. Then we verify π^{Real} using Lemma 3.8, reject immediately if π^{Real} did not pass the test. If π^{Real} passes the test (which means that it is "close" to a Boolean proof π^{Bool}), we use the algorithm described above to estimate p_{acc}. We accept x if and only if our estimation is above $2/3$.

The correctness of M^{PCPP} is easy to see:

Claim 3.11. *For every input x, if $x \notin L$ then M^{PCPP} rejects x; while if $x \in L$ and x has an easy witness then M^{PCPP} accepts x.*

Proof. If $x \notin L$, then it always holds that $p_{\mathsf{acc}} < 1/2$, so M^{PCPP} rejects. If $x \in L$ and x has an easy witness, then there exists a proof π, $(w_1, \ldots, w_{H_{\mathsf{proof}}})$ and $(\alpha_1, \ldots, \alpha_{H_{\mathsf{proof}}})$ such that

1. for every $j \in [W_{\mathsf{proof}}]$, $\pi^{\mathsf{Real}}_{i,j} \in [0,1]$;
2. for every $i \in [W_{\mathsf{proof}}], j \in [H_{\mathsf{proof}}]$, $\|\pi^{\mathsf{Real}}_i - \pi_i\|_1 \leq \delta$.

Note that $\|\pi^{\mathsf{Real}}_i - \pi^{\mathsf{Bool}}_i\|_1 \leq \|\pi^{\mathsf{Real}}_i - \pi_i\|_1 \leq \delta$ since π^{Bool}_i is the closest Boolean string to π^{Real}_i, and thus $\|\pi_i - \pi^{\mathsf{Bool}}_i\|_1 \leq 2\delta$. Since the probability that VPCPP accepts π is 1, by Lemma 2.11, $p_{\mathsf{acc}} \geq 1 - q \cdot 2\delta > 5/6$, so M^{PCPP} accepts. □

The machine M^{PCPP} guesses $H_{\mathsf{proof}}(n+a) < n_{\mathsf{hard}}/10$ bits of nondeterminism, and uses $\tilde{O}(s\ell) < n_{\mathsf{hard}}/10$ bits of advice. Thus $L(M^{\mathsf{PCPP}}) \in \mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T/\log^{c_{\mathsf{hard}}}(T), n_{\mathsf{hard}}/10]_{/(n_{\mathsf{hard}}/10)}$.

3.2.6 The $\mathsf{FP}^{\mathsf{NP}}$ Algorithm for Remote-Point

Let $C : \{0,1\}^n \to \{0,1\}^\ell$ be the input circuit. We first construct the hard language L^{hard} and the algorithm M^{PCPP}. Since M^{PCPP} is a nondeterministic RAM algorithm that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, uses at most $n_{\mathsf{hard}}/10$ nondeterministic bits and at most $n_{\mathsf{hard}}/10$ advice bits, it follows that there is an input $x_{\mathsf{hard}} \in \{0,1\}^{n_{\mathsf{hard}}}$ such that $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) \neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$. Moreover, let α be the advice string fed to M^{PCPP}, i.e., the circuit C. We can find such an input x_{hard} by running $\mathcal{R}(1^{n_{\mathsf{hard}}}, M^{\mathsf{PCPP}}, \alpha)$, where $\mathcal{R}$ is the refuter guaranteed by Theorem 2.2. Thus, we can find x_{hard} in deterministic $\mathrm{poly}(T)$ time with an NP oracle.

It follows from Claim 3.11 that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but x_{hard} does not have an easy witness. Thus, we can use the NP oracle to find the lexicographically first PCPP proof matrix π such that

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

Then, there must exist a row π_i that is $(1/2-\varepsilon)$-far from $\mathsf{Range}(C)$. To see this, suppose that for every i, the i-th row π_i is $(1/2-\varepsilon)$-close to $\mathsf{Range}(C)$. Then there exists some $w_i \in \{0,1\}^n$ such that $\delta(\mathsf{Amp}(\pi_i), C(w_i)) \leq 1/2-\varepsilon$. By Theorem 2.5, there is an advice α_i such that $\mathsf{dec}_{\alpha_i}(C(w_i))$ satisfies (1) for every $j \in [W_{\mathsf{proof}}]$, $(\mathsf{dec}_{\alpha_i}(C(w_i)))_j \in [0,1]$; (2) $\|\mathsf{dec}_{\alpha_i}(C(w_i)) - \pi_i\|_1 \leq \delta$. It follows that π is an easy witness for x_{hard}, a contradiction.

Finally, we use the NP oracle to find the first row π_i, such that $\mathsf{Amp}(\pi_i)$ is $(1/2-\varepsilon)$-far from $\mathsf{Range}(C)$. The overall procedure takes deterministic $\mathrm{poly}(T) \leq \mathrm{poly}(\ell)$ time with an NP oracle.

3.3 Variants of Our Frameworks

In this sub-section, we discuss two variants of our frameworks in Theorems 3.2 and 3.5.

Allowing $\mathsf{FP}^{\mathsf{NP}}$ preprocessing in Theorem 3.5. Notice that in Theorem 3.2, we allow the non-trivial algorithm for Satisfying-Pairs to have a polynomial-time preprocessing with an NP oracle on the circuits before seeing the inputs. In our framework for remote point problems (see Theorem 3.5), we did not consider the preprocessing phase, as we do not need any preprocessing in our unconditional results, and the trade-off between parameters will be too complicated. We now argue informally that by modifying our framework as follows, it suffices to have algorithms for Satisfying-Pairs with an $\mathsf{FP}^{\mathsf{NP}}$ preprocessing phase that generates a "short" data structure.

- Fix any integer d. Recall that Theorem 3.7 states that non-trivial algorithms for (approximate counting version, omitted below) $\mathsf{AND}_d \circ \mathscr{C}$-Satisfying-Pairs imply non-trivial algorithms for $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$-Satisfying-Pairs. By verifying the proof of Theorem 3.7, we can show that if the $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit has fan-in A, the algorithm will call $\mathsf{AND}_d \circ \mathscr{C}$-Satisfying-Pairs for A^d times. Moreover, if the $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit is fixed, then the circuit parts of these A^d instances for $\mathsf{AND}_d \circ \mathscr{C}$-Satisfying-Pairs are also fixed.

- In the proof of Theorem 3.5, we need to apply the non-trivial algorithm for $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$-Satisfying-Pairs on three types of circuits: circuits C^{norm} and C^{diff} in Lemma 3.8 (see Appendix A.2), and circuit C^{Prod} in Section 3.2.4. There are $O(2^{r_{\mathsf{shared}}}) \leq O(\ell^{1/m})$ circuits in total, all of which are constructible given the input circuit $C : \{0,1\}^n \to \{0,1\}^\ell$. Moreover, the fan-in of these $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$-Satisfying-Pairs circuits are $O(\log \ell)$, and $d = O(1)$. Therefore we will need to call the $\mathsf{AND}_d \circ \mathscr{C}$-Satisfying-Pairs for $O(\ell^{1/m}) \cdot (\log \ell)^d = \ell^{o(1)}$ times, and the circuits are fixed given the input circuit C. (That is, the circuits do not depend on the input of the machine M^{PCPP}.)

- Now we assume that our non-trivial algorithm for $\mathsf{AND}_d \circ \mathscr{C}$-Satisfying-Pairs has an $\mathsf{FP}^{\mathsf{NP}}$ preprocessing phase. Similar to the proof of Theorem 3.2, the final $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}$-Remote-Point works as follows. Given any input circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, we construct the circuits in all $\ell^{o(1)}$ instances of $\mathsf{AND}_d \circ \mathscr{C}$-Satisfying-Pairs that will be needed in the proof, run the $\mathsf{FP}^{\mathsf{NP}}$ preprocessing on these instances, and put the generated data structure into the advice of M^{PCPP}. It remains to verify that there are enough space to store these data structures in the advice. The non-deterministic time hierarchy theorem allows us to have $n_{\mathsf{hard}}/10$ bits of advice (see Theorem 2.2), where $n_{\mathsf{hard}} = 20H_{\mathsf{proof}} \cdot n \geq \ell^{5q+1}$. The circuit C of size $s \cdot \ell$, which can be properly encoded using $\tilde{O}(s \cdot \ell) \leq \ell^2$ bits, needs to be put in the advice; therefore the rest $n_{\mathsf{hard}}/10 - \ell^2$ bits in advice can be used to store the data structure generated in the preprocessing phase.

Algorithm for "weak" remote point. We can see that Theorem 3.5 requires the input circuit C to have quasi-polynomial stretch. This is because the hardness amplification procedure Amp (i.e. the XOR lemma with linear-sum decoder, see Section 2.4 and Appendix A.5) requires quasi-polynomial stretch. An interesting open problem is then to improve the amplification procedure to achieve a better stretch in Theorem 3.5.

The application of the hardness amplification procedure is to reduce the "strong" remote point (i.e. the distance is $1/2 - n^{\Omega(1)}$) to "weak" remote point (i.e. the distance is $\Omega(1)$). If we only want to have an $\mathsf{FP}^{\mathsf{NP}}$ algorithm in the "weak" remote point regime, it suffices to follow the proof of Theorem 3.2 and replace the rectangular PCPP to the smooth and rectangular PCPP in Theorem 2.14, which can deal with $\mathscr{C}$-Remote-Point with polynomial stretch. (The key is that smooth PCPP can tolerate errors in the proof, see Lemma 2.11 and Section 3.2.5).

4 Hard Partial Truth Tables

In this section, we employ the same technique to construct algorithms for Partial-Hard and Partial-AvgHard from non-trivial Satisfying-Pairs algorithms. We only consider circuit classes that are both *typical* and *complete*. Recall that a circuit class $\mathscr{C}$ is typical if it is closed under negations and projections. We say $\mathscr{C}$ is *complete* if given a truth table tt of length 2^n, we can compute a $\mathscr{C}$ circuit of size $2^{O(n)}$ whose truth table is tt in deterministic $2^{O(n)}$ time.

4.1 Hard Partial Truth Tables from Satisfying-Pairs

Instead of allowing a P^{NP} preprocessing on the circuits, the algorithm for Satisfying-Pairs used to solve Partial-Hard allows a P^{NP} preprocessing on *inputs*, formally defined as follows.

Definition 4.1 (Algorithms for Satisfying-Pairs with P^{NP} Preprocessing on Inputs)**.** Let P be one of the problems $\mathscr{C}$-Satisfying-Pairs, $\#\mathscr{C}$-Satisfying-Pairs, Approx_δ-$\mathscr{C}$-Satisfying-Pairs, Gap_δ-$\mathscr{C}$-Satisfying-Pairs. A t-time algorithm for P with P^{NP} preprocessing of an ℓ-size data structure on inputs is a pair of algorithms (A_1, A_2) that solves P in two phases:

1. Given the inputs $x_1, x_2, \ldots, x_M \in \{0,1\}^n$, the polynomial-time algorithm A_1 with oracle access to a SAT oracle computes a string $\mathsf{DS} \in \{0,1\}^\ell$.
2. Given the circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s and the string DS, the algorithm A_2 solves P on the instance $(C_1, \ldots, C_N, x_1, \ldots, x_M)$ in time t.

Theorem 4.2. *There are constants $\varepsilon > 0$ and c_{imp} such that the following holds. Let $0 < \eta < 1/2$ be a constant, $\mathscr{C}[s]$ be a typical and complete circuit class where $s = s(n) > n$ is a size parameter, and $\mathscr{C}'[2s] := \mathsf{OR}_2 \circ \mathscr{C}[s]$. Let $\ell(n)$ be a good function such that $s(n)^{1+\Omega(1)} \le \ell(n) \le 2^n$.*

Assumption: *Suppose that for some constant $c \ge 1$, there is an $(NM/\log^{c_{\mathsf{imp}}}(NM))$-time algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs with $N := \ell^{c+1-\eta} \cdot \mathrm{polylog}(\ell)$ circuits of size $\mathrm{poly}(s(n))$ and $M := \ell^{1-\eta} \cdot \mathrm{polylog}(\ell)$ inputs of length $2n$, allowing a P^{NP} preprocessing of an M^c-size data structure on inputs.*

Conclusion: *There is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Partial-Hard with $\ell(n)$ input strings. More precisely, given a list of inputs $z_1, z_2, \dots, z_\ell \in \{0,1\}^n$, we can compute a list of bits $b_1, b_2, \dots, b_\ell$ such that for every $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}$ of size s, there exists an $i \in [\ell]$ such that $C(z_i) \ne b_i$.*

Proof Sketch of Theorem 4.2. The proof is very similar to the proof of Theorem 3.2; in fact, it is (nearly) equivalent to first reducing Partial-Hard to Avoid and then invoking Theorem 3.2. Therefore we only highlight the differences.

It is without loss of generality to assume ℓ is a power of 2 and $c \ge 2$. We set the following parameters:

$$\begin{aligned}
m &:= 5(c+2)/\eta = O(1), \\
w_{\mathsf{proof}} &:= \log \ell, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = \ell, \\
h_{\mathsf{proof}} &:= (c+1)\log \ell, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = \ell^{c+1}, \\
n_{\mathsf{hard}} &:= 100 H_{\mathsf{proof}} \cdot s \log s, \\
T &:= H_{\mathsf{proof}} \cdot W_{\mathsf{proof}} / \log^{c_{\mathsf{tm}}}(\ell), \\
h_{\mathsf{input}} &:= \left(1 - \tfrac{\Theta(\log\log T)}{\log T}\right) h_{\mathsf{proof}}, & H_{\mathsf{input}} &:= 2^{h_{\mathsf{input}}} = H_{\mathsf{proof}}/\mathrm{polylog}(\ell), \\
w_{\mathsf{input}} &:= \lceil \log \tilde{n}_{\mathsf{hard}} \rceil - h_{\mathsf{input}}, & W_{\mathsf{input}} &:= 2^{w_{\mathsf{input}}} = s \log s \cdot \mathrm{polylog}(\ell).
\end{aligned}$$

Here $\tilde{n}_{\mathsf{hard}} = O(n_{\mathsf{hard}})$ is the codeword length of a length-n_{hard} string encoded via Enc where $(\mathsf{Enc}, \mathsf{Dec})$ is a fixed error-correcting code in Theorem 2.1; and c_{tm} is a sufficiently large constant.

We can check the technical condition $n_{\mathsf{hard}}^{1+\Omega(1)} \le T \le 2^{\mathrm{poly}(n_{\mathsf{hard}})}$, so it is valid to invoke Theorem 2.2. Also, $(5/m)\log T \le w_{\mathsf{proof}}$, so it is valid to invoke the 2-query rectangular PCPP in Theorem 2.13. There are other checks for technical conditions that we omit here.

The first difference is the definition of "easy witness". We say *x has an easy witness* if there is a proof matrix π (of size $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$) for the statement "$\mathsf{Enc}(x) \in L^{\mathsf{enc}}$" such that:

(completeness) for every $\mathsf{seed} \in \{0,1\}^r$, $\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})$ accepts with probability at least c_{pcp};

(easiness) for every row π_j of π, there exists a *size-s $\mathscr{C}$ circuit* $C_j : \{0,1\}^n \to \{0,1\}$ such that for every i, $\pi_{j,i} = C_j(z_i)$.

Then, our machine M^{PCPP} guesses H_{proof} size-s $\mathscr{C}$ circuits $C_1, C_2, \dots, C_{H_{\mathsf{proof}}} : \{0,1\}^n \to \{0,1\}$. Let π be the $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ proof matrix where for each $j \in [H_{\mathsf{proof}}], i \in [W_{\mathsf{proof}}]$, $\pi_{j,i} = C_j(z_i)$. We need to estimate

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

We still reduce the problem of estimating p_{acc} to $2^{r_{\mathsf{shared}}}$ instances of $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs, where $\varepsilon := (c_{\mathsf{pcp}} - s_{\mathsf{pcp}})/4$. However, now, each instance consists of $N := 2^{r_{\mathsf{row}}} = 2^{h_{\mathsf{proof}} - (5/m)\log T}$ circuits and $M := 2^{r_{\mathsf{col}}} = 2^{w_{\mathsf{proof}} - (5/m)\log T}$ inputs.[18]

[18] That is, the role of inputs and circuits are swapped as compared to Theorem 3.2.

We enumerate $\mathsf{seed.shared}$. For each $\mathsf{seed.shared}$, we create an $\mathsf{Approx}_\varepsilon$-$\mathscr{C}'$-Satisfying-Pairs instance $\mathcal{I}_{\mathsf{seed.shared}}$ corresponding to $\mathsf{seed.shared}$, which contains an input $\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.col}}$ for every $\mathsf{seed.col}$ and a circuit $C_{\mathsf{seed.shared},\mathsf{seed.row}}$ for every $\mathsf{seed.row}$. We elaborate on how this instance is constructed as this is different from Theorem 3.2.

Each $\mathsf{seed.col}$ corresponds to an input $\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.col}}$ of the following form:

$$(a_1, \ldots, a_q, pc_1^{\mathsf{col}}, \ldots, pc_p^{\mathsf{col}}),$$

where $p + q \leq 2$, for each $i \in [q]$,

$$a_i := \begin{cases} \mathsf{icol}[i] & \text{if } \mathsf{itype}[i] = \mathsf{input}; \\ z_{\mathsf{icol}[i]} & \text{if } \mathsf{itype}[i] = \mathsf{proof}, \end{cases}$$

and pc_i^{col} represents the contribution of $\mathsf{seed.col}$ in the i-th parity-check bit.

The circuit $C_{\mathsf{seed.shared},\mathsf{seed.row}}$ corresponding to $\mathsf{seed.row}$ is as follows:

- It receives input $(a_1, \ldots, a_q, pc_1^{\mathsf{col}}, \ldots, pc_p^{\mathsf{col}})$.
- For each $j \in [q]$, let
$$\mathsf{ans}_j = \begin{cases} \mathsf{Enc}(x)_{\mathsf{irow}[j],a_j} & \text{if } \mathsf{itype}[j] = \mathsf{input} \\ C_{\mathsf{irow}[j]}(a_j) & \text{if } \mathsf{itype}[j] = \mathsf{proof} \end{cases}.$$
Note that since $\mathscr{C}$ is complete, we can compute a $\mathscr{C}$ circuit of size $\mathrm{poly}(W_{\mathsf{input}}) = \mathrm{poly}(s)$ whose truth table is the $\mathsf{irow}[j]$-th row of $\mathsf{Enc}(x)$. That is, we can compute a $\mathscr{C}$ circuit of size $\mathrm{poly}(s)$ that on input a_j, outputs ans_j.
- For each $j \in [q]$, let pc_j^{row} be the contribution of $\mathsf{seed.row}$ in the j-th parity-check bit.
- It returns
$$\mathsf{VDec}(\mathsf{ans}_1, \ldots, \mathsf{ans}_q, pc_1^{\mathsf{col}} \oplus pc_1^{\mathsf{row}}, \ldots, pc_p^{\mathsf{col}} \oplus pc_p^{\mathsf{row}}).$$

Here, VDec is the decision predicate of VPCPP, and is an OR_2 of its input bits or their negations. Since $\mathscr{C}$ is typical, C is a $\mathsf{OR}_2 \circ \mathscr{C}$ circuit. And one can easily verify that for each $\mathsf{seed} = (\mathsf{seed.shared}, \mathsf{seed.row}, \mathsf{seed.col})$, $C_{\mathsf{seed.shared},\mathsf{seed.row}}(\mathsf{Input}_{\mathsf{seed.shared},\mathsf{seed.col}}) = 1$ if and only if $\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})$ accepts. It follows that we can estimate p_{acc} by solving the instances $\mathcal{I}_{\mathsf{seed.shared}}$ for every $\mathsf{seed.shared}$.

To summarise, our algorithm M^{PCPP} works as follows. It first computes $\mathsf{Enc}(x)$ and guesses $C_1, C_2, \ldots, C_{H_{\mathsf{proof}}}$. Then, it enumerates $\mathsf{seed.shared}$, produces the instances $\mathcal{I}_{\mathsf{seed.shared}}$, and feeds them to the algorithm for $\mathsf{Approx}_\varepsilon$-$\mathscr{C}$-Satisfying-Pairs to obtain an estimation $p'_{\mathsf{acc}}(\mathsf{seed.shared})$. Let p'_{acc} be the average of $p'_{\mathsf{acc}}(\mathsf{seed.shared})$ over all $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$.

We can still see that M^{PCPP} rejects every $x \notin L^{\mathsf{hard}}$ and accepts every x with an easy witness. The machine M^{PCPP} runs in $T/\log^{c_{\mathsf{hard}}} T$ time, guesses $H_{\mathsf{proof}} \cdot 5s \log s < n_{\mathsf{hard}}/10$ nondeterministic bits (since a size-s circuit can be encoded with at most $5s \log s$ bits), and uses at most $\ell^{c+1} < n_{\mathsf{hard}}/10$ advice bits. By Theorem 2.2, M^{PCPP} cannot compute L^{hard}.

The hard partial truth table algorithm. Given a list of inputs $z_1, z_2, \ldots, z_\ell \in \{0,1\}^n$, our algorithm for finding a hard partial truth table $((z_1, b_1), (z_2, b_2), \ldots, (z_\ell, b_\ell))$ works as follows. First, we construct the hard language L^{hard} and the algorithm M^{PCPP}. Let α be the advice string fed to M^{PCPP} and $\mathcal{R}$ be the refuter in Theorem 2.2, we can use $\mathcal{R}(1^{n_{\mathsf{hard}}}, M^{\mathsf{PCPP}}, \alpha)$ to find an input

x_{hard} where M^{PCPP} fails on x_{hard}; in particular, $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) = 0$ but $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$. This takes deterministic $\mathrm{poly}(H_{\mathsf{proof}}) = \mathrm{poly}(\ell)$ time with an NP oracle.

Then we find the lexicographically first proof matrix π such that $\mathsf{VPCPP}^{\mathsf{Enc}(x_{\mathsf{hard}})\circ\pi}$ accepts w.p. at least c_{pcp}, using the NP oracle. There has to be some $j \in [H_{\mathsf{proof}}]$ such that for every size-s $\mathscr{C}$ circuit C, there exists $i \in [W_{\mathsf{proof}}]$ such that $C(z_i) \neq \pi_{j,i}$; moreover, the first such j can be found in $\mathrm{poly}(H_{\mathsf{proof}}) = \mathrm{poly}(\ell)$ time with an NP oracle. We can pick

$$((z_1, \pi_{j,1}), (z_2, \pi_{j,2}), \ldots, (z_{W_{\mathsf{proof}}}, \pi_{j,W_{\mathsf{proof}}}))$$

as the partial truth table hard for size-s $\mathscr{C}$ circuits. □

4.2 Average-Case Hard Partial Truth Tables

Theorem 4.3. *There is a universal constant $c_u \geq 1$ such that the following holds. Let $s = s(n) > n$ be a circuit size parameter, $N := N(n)$ be a parameter such that $2^{\log^{c_u} s} < N < 2^{s^{0.99}}$, $\varepsilon := \varepsilon(n) > s^{-c_u}$ be the error parameter, and $\ell := N^{c_u \log(1/\varepsilon)}$. Let $\mathscr{C}[s]$ be a typical and complete circuit class, and denote $\mathscr{C}'[c_u s] := \mathsf{AND}_{c_u} \circ \mathscr{C}[s]$ (i.e. a $\mathscr{C}'$ circuit of size $c_u s$ refers to the AND of at most c_u $\mathscr{C}$ circuits of size s).*

Assumption: *Let $P := (\log N)^{\log(1/\varepsilon)}$. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} := N^2/P^{c_u}$ that, given as input a list of N $\mathscr{C}'[c_u s]$ circuits $\{C_i\}$ and a list of N inputs $\{x_j\}$ with input length $n \cdot \mathrm{polylog}(\ell)$, estimates $\Pr_{i,j\leftarrow[N]}[C_i(x_j)]$ with additive error $\eta := \varepsilon^{c_u}$.*

Conclusion: *There is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathscr{C}[s]$-Partial-AvgHard with $\ell(n)$ input strings. More precisely, given a list of inputs $w_1, w_2, \ldots, w_\ell \in \{0,1\}^n$, we can compute a list of bits $b_1, b_2, \ldots, b_\ell$ such that for every $\mathscr{C}$ circuit $C : \{0,1\}^n \to \{0,1\}$ of size s,*

$$\Pr_{i\leftarrow[\ell]}[C(w_i) \neq b_i] \geq \frac{1}{2} - \varepsilon.$$

Proof Sketch of Theorem 4.3. The proof is similar to that of Theorem 3.5, so here we only highlight the difference. Roughly speaking, the main difference is that we swap the role of inputs and circuits.

For a circuit C and a list of inputs $w_1, w_2, \ldots, w_\ell$, with slight abuse of notation, we define $C(w) := C(w_1) \circ C(w_2) \circ \cdots \circ C(w_\ell)$.

Analysing $\mathsf{Prod} \circ \mathsf{Sum}$ circuits. Let $d \geq 1$ be a constant. We use $\mathsf{Prod} \circ \mathsf{Sum}$ to denote the class of multi-output circuits that take inputs $y \in \{0,1\}^{\ell_y}$ and α, and has the following components:

- Let ℓ_{Sum} denote the number of middle "linear sum" gates. For each $i \in [\ell_{\mathsf{Sum}}]$, the i-th gate outputs
$$\mathsf{Sum}_i(y, \alpha) := \sum_{k\in[A]} \mathsf{coeff}_k(\alpha) \cdot y_{\mathsf{idx}_k(\alpha,i)}.$$
- Let ℓ_{Prod} denote the number of output gates. The i-th output gate is a product gate of fan-in d, and is connected to the $q_1(i), q_2(i), \ldots, q_d(i)$-th linear sum circuits. Its output is
$$C_i^{\mathsf{Prod}}(y, \alpha) := \prod_{t=1}^{d} \mathsf{Sum}_{q_t(i)}(y, \alpha).$$

Remark 4.4. The important measures of a $\mathsf{Prod}_d \circ \mathsf{Sum}$ circuit are:

- the number of gates in each level (ℓ_{Sum}, ℓ_{Prod});
- the fan-in of the top Prod gates (d);
- the fan-in (A), coefficient sum (U), and locality (l) of the linear sum layer.

It turns out that as an intermediate step, we need an algorithm provided by the following lemma.

Lemma 4.5. *Let $\mathscr{C}$ be a typical circuit class, $M' \geq 1$ and $\eta \in (0,1)$ be parameters. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} = T^{\mathrm{alg}}(N, M)$ that, given as input a list of $\hat{M} \leq M$ $\mathsf{AND}_d \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of $\hat{N} \leq N$ inputs $\{x_j\}$ of length $n \cdot \mathrm{polylog}(\ell)$, estimates the following quantity with additive error η:*

$$\Pr_{i \leftarrow [\hat{M}], j \leftarrow [\hat{N}]}[C_i(x_j)].$$

Then, for any constant $\ell_{\mathscr{C}} > 0$, there is a deterministic algorithm running in time $A^d \cdot (\ell_{\mathscr{C}}^d + \ell_{\mathsf{Prod}}/N) \cdot (2^{dl} + M'/M) \cdot O(T^{\mathrm{alg}})$ that, given as input a $\mathsf{Prod}_d \circ \mathsf{Sum}$ circuit C^{Prod} with parameters specified in Remark 4.4, a list of ℓ_x strings $\{x_j\}$, a list of M' inputs $\{\alpha_j\}$, and a list of M' $\mathscr{C}$ circuits $\{C_j\}$ from $\{0,1\}^{|x|}$ to $\{0,1\}^{\ell_{\mathscr{C}}}$, estimates the following quantity with additive error $\eta \cdot U^d$:

$$\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow [M']} \left[C_i^{\mathsf{Prod}}(C_j(x), \alpha_j) \right].$$

Recall here that $C_j(x) = C_j(x_1) \circ C_j(x_2) \circ \cdots \circ C_j(x_{\ell_x})$.

The proof is similar to that of Theorem 3.7 and is deferred to Appendix A.1.

Set up. We set the parameters as follows.

$$\begin{aligned}
\delta &:= (10^9 q)^{-q^2}, \\
m &:= c_m \log(1/\varepsilon)/\delta, \\
w_{\mathsf{proof}} &:= (60q/m)\log \ell, & W_{\mathsf{proof}} &:= 2^{w_{\mathsf{proof}}} = \ell^{O(\delta/\log(1/\varepsilon))}, \\
h_{\mathsf{proof}} &:= (5q+1)\log \ell, & H_{\mathsf{proof}} &:= 2^{h_{\mathsf{proof}}} = \ell^{5q+1}, \\
n_{\mathsf{hard}} &:= 200 H_{\mathsf{proof}} \cdot s \log s, \\
T &:= H_{\mathsf{proof}} \cdot W_{\mathsf{proof}} / \log^{c_{\mathsf{tm}}}(\ell). \\
h_{\mathsf{input}} &:= \left(1 - \frac{\Theta(m^2 \log\log T)}{\log T}\right) h_{\mathsf{proof}}, & H_{\mathsf{input}} &:= 2^{h_{\mathsf{input}}} = H_{\mathsf{proof}}/\mathrm{polylog}(\ell), \\
w_{\mathsf{input}} &:= \lceil \log \tilde{n}_{\mathsf{hard}} \rceil - h_{\mathsf{input}}, & W_{\mathsf{input}} &:= 2^{w_{\mathsf{input}}} = s \log s \cdot \mathrm{polylog}(\ell). \\
a &:= O(\log^2 W_{\mathsf{proof}}/(\varepsilon\delta)^2) = O(\log^2 \ell/\varepsilon^2), \\
A &:= O(\log W_{\mathsf{proof}}/(\varepsilon\delta)^2) = O(\log \ell/\varepsilon^2), \\
U &:= O(1/\varepsilon), \\
l &:= \log \ell.
\end{aligned}$$

Here c_m and c_{tm} are sufficiently large constants, q is the query complexity of the smooth and rectangular PCPP in Theorem 2.14, and $\tilde{n}_{\mathsf{hard}} = \Theta(n_{\mathsf{hard}})$ is the length of $\mathsf{Enc}(x)$ when the length of x is n_{hard}. Let $\hat{H}_{\mathsf{proof}}$ be the number of rows of the PCPP proof in Theorem 2.14, and let $\hat{h}_{\mathsf{proof}} = \log \hat{H}_{\mathsf{proof}}$. Also let $r, r_{\mathsf{shared}}, r_{\mathsf{col}}, r_{\mathsf{row}}$ be the total, shared, column, row randomness in Theorem 2.14 respectively.

We use a different definition of "easy witness " as follows. We say x has an easy witness if there is a proof matrix π such that:

(completeness) for every $\mathsf{seed} \in \{0,1\}^r$, $\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi}(\mathsf{seed})$ accepts;

(*approximate* easiness) for every row π_i of π, there exists a size-s $\mathscr{C}$ circuit $C_i : \{0,1\}^n \to \{0,1\}$ and an advice $\alpha_i \in \{0,1\}^a$ such that the decoding of the string $C_i(w)$ with advice α_i is δ-close to π_i with respect to ℓ_1-norm. (Recall that $w = (w_1, w_2, \ldots, w_\ell)$ is our input and $C(w)$ denotes the concatenation of $C(w_1)$, $C(w_2)$, ..., $C(w_\ell)$.) In particular:

1. for every $j \in [W_{\mathsf{proof}}]$, $(\mathsf{dec}_{\alpha_i}(C_i(w)))_j \in [0,1]$;
2. $\|\mathsf{dec}_{\alpha_i}(C_i(w)) - \pi_i\|_1 \le \delta$.

Our machine guesses $\hat{H}_{\mathsf{proof}}$ size-s $\mathscr{C}$ circuits $C_1, C_2, \ldots, C_{\hat{H}_{\mathsf{proof}}} : \{0,1\}^n \to \{0,1\}$ as well as $\hat{H}_{\mathsf{proof}}$ advices $\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}}$. Let $\pi_i^{\mathsf{Real}} := \mathsf{dec}_{\alpha_i}(C_i(w))$, and π_i^{Bool} be the Boolean string that is closest to π_i^{Real}. For $\iota \in [q]$, we define

$$f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota}(\mathsf{seed.row}, \mathsf{seed.col}) = \pi^{\mathsf{Bool}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} \quad \text{and}$$
$$f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}(\mathsf{seed.row}, \mathsf{seed.col}) = \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}.$$

Verifying closeness of π^{Bool} and π^{Real}. The next lemma shows that we can verify whether a Boolean proof π^{Bool} and a real proof π^{Real} are close. The proof is deferred to Appendix A.3.

Lemma 4.6. *Let $\mathscr{C}$ be a typical circuit class and $d \ge 2$ be an even number. Suppose there is an algorithm that takes as inputs a list of $2^{r_{\mathsf{col}}}$ $\mathsf{AND}_{2d} \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of $2^{r_{\mathsf{col}}}$ inputs $\{x_j\}$ of length $n \cdot \mathrm{polylog}(\ell)$, runs in deterministic T^{alg} time, and estimates the following quantity with additive error η:*

$$\Pr_{i,j \leftarrow [2^{r_{\mathsf{col}}}]}[C_i(x_j)].$$

Then there is an algorithm that takes the strings $w_1, w_2, \ldots, w_\ell$, circuits $(C_1, C_2, \ldots, C_{\hat{H}_{\mathsf{proof}}})$, and $(\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}})$ as inputs, runs in deterministic $O((3A)^{2d} T^{\mathrm{alg}}) \cdot (2^{2dl + r_{\mathsf{shared}}} + T \log^{O(m)} T / 2^{2r_{\mathsf{col}}}))$ time, and satisfies the following:

(Completeness) *If for every $i \in [\hat{H}_{\mathsf{proof}}]$, it holds that (1) for every $j \in [W_{\mathsf{proof}}]$, $\pi^{\mathsf{Real}}_{i,j} \in [0,1]$; (2) $\|\pi_i^{\mathsf{Real}} - \pi_i^{\mathsf{Bool}}\|_1 \le \delta$, then the algorithm accepts.*

(Soundness) *If the algorithm accepts, then it holds that*

1. *for every $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ and $\iota \in [q]$, $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d \le 1 + 2\eta \cdot U^d$;*
2. $\mathbb{E}_{i \leftarrow [\hat{H}_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]} \left[|\pi^{\mathsf{Real}}_{i,j} - \pi^{\mathsf{Bool}}_{i,j}|^d \right] \le 4^d \cdot \delta + 2^{d+1} \eta (2U+1)^{2d}$.

Estimating p_{acc}. Now we verified that π^{Real} is close to π^{Bool} using Lemma 4.6, with parameter $d = 2q$. After that, the next step is to use it to speed up L^{hard}. We estimate

$$p_{\mathsf{acc}} := \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r} \left[\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi^{\mathsf{Bool}}}(\mathsf{seed}) \text{ accepts} \right].$$

Actually, it suffices to distinguish between the case that $p_{\mathsf{acc}} > 5/6$ and the case that $p_{\mathsf{acc}} < 1/2$.

We still enumerate $\mathsf{seed.shared}$, and we now need to estimate

$$p_{\mathsf{acc}}(\mathsf{seed.shared}) := \Pr_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}}\left[\mathsf{VPCPP}^{\mathsf{Enc}(x)\circ\pi^{\mathsf{Bool}}}(\mathsf{seed})\text{ accepts}\right].$$

Let $pc_1, pc_2, \ldots, pc_q \leftarrow V_{\mathsf{pc}}(\mathsf{seed.shared})$ be the parity-check bits of the PCPP verifier, and let pc_ι^{row} (resp. pc_ι^{col}) denote the contribution of $\mathsf{seed.row}$ (resp. $\mathsf{seed.col}$) to pc_ι, then $pc_\iota = pc_\iota^{\mathsf{row}} \oplus pc_\iota^{\mathsf{col}}$.

As in the proof of Theorem 3.5, here it suffices to estimate for every $S, S' \subseteq [q]$

$$p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S') = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}}\left[\prod_{\iota\in S} a_\iota^{\mathsf{Real}} \cdot \prod_{\iota\in S'} pc_\iota\right],$$

where

$$a_\iota^{\mathsf{Real}} := \begin{cases} \tilde{x}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} & \mathsf{itype}[\iota] = \mathsf{input},\\ \pi^{\mathsf{Real}}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} & \mathsf{itype}[\iota] = \mathsf{proof}. \end{cases}$$

We want to invoke Lemma 4.5 to estimate this, so we want to construct a $2^{r_{\mathsf{col}}}$-output $\mathsf{Prod}\circ\mathsf{Sum}$ circuit C^{Prod}, $2^{r_{\mathsf{row}}}$ circuits $\{C_{\mathsf{seed.row}}\}$ and $2^{r_{\mathsf{row}}}$ strings $\{\alpha_{\mathsf{seed.row}}\}$ such that

$$\begin{aligned}
&C^{\mathsf{Prod}}_{\mathsf{seed.col}}(C_{\mathsf{seed.row}}(w), \alpha_{\mathsf{seed.row}})\\
&= \prod_{\iota\in S} a_\iota^{\mathsf{Real}} \cdot \prod_{\iota\in S'} pc_\iota\\
&= \prod_{\iota\in S^{\mathsf{proof}}}\left(\sum_{k\in[A]} \mathsf{coeff}_k(\alpha_{\mathsf{irow}[\iota]}) \cdot C_{\mathsf{irow}[\iota]}\left(w_{\mathsf{idx}_k(\alpha_{\mathsf{irow}[\iota]},\mathsf{icol}[\iota])}\right)\right) \cdot \prod_{\iota\in S^{\mathsf{input}}} \tilde{x}_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]} \cdot \prod_{\iota\in S'}(pc_\iota^{\mathsf{row}} \oplus pc_\iota^{\mathsf{col}})
\end{aligned} \tag{6}$$

where $S^{\mathsf{proof}} = \{\iota \in S : \mathsf{itype}[\iota] = \mathsf{proof}\}$ and $S^{\mathsf{input}} = \{\iota \in S : \mathsf{itype}[\iota] = \mathsf{input}\}$. This motivates the following definitions.

For $i \in [\ell]$, let $z_i \in \{0,1\}^{n+w_{\mathsf{input}}}$ be the string such that the first n bits of z_i is w_i, and the last w_{input} bit is

$$\begin{cases} i & i \in [W_{\mathsf{input}}],\\ 1 & i \notin [W_{\mathsf{input}}]. \end{cases}$$

Here we identify $[W_{\mathsf{input}}]$ with $\{0,1\}^{w_{\mathsf{input}}}$.

For any string v, define $\hat{C}_v : \{0,1\}^{\lceil\log|v|\rceil} \to \{0,1\}$ as the circuit that on input $i \le |v|$, outputs v_i. Since $\mathscr{C}$ is complete, $\hat{C}_v$ is an efficiently computable $\mathscr{C}$ circuit of size $\mathrm{poly}(|v|)$. Let $\mathsf{proj}_{\mathsf{input}} : \{0,1\}^{n+w_{\mathsf{input}}} \to \{0,1\}^{w_{\mathsf{input}}}$ be the circuit that outputs the last w_{input} bits of its input, and let $\mathsf{proj}_{\mathsf{proof}} : \{0,1\}^{n+w_{\mathsf{input}}} \to \{0,1\}^n$ be the circuit that outputs the first n bits of its input.

Now for fixed $\mathsf{seed.row}$, define

$$C_\iota^a := \begin{cases} \hat{C}_{\tilde{x}_{\mathsf{irow}[\iota]}} \circ \mathsf{proj}_{\mathsf{input}} & \mathsf{itype}[\iota] = \mathsf{input}\\ C_{\mathsf{irow}[\iota]} \circ \mathsf{proj}_{\mathsf{proof}} & \mathsf{itype}[\iota] = \mathsf{proof} \end{cases}$$

for $\iota \in S$, and C_ι^{pc} be the circuit that outputs pc_ι^{row} for $\iota \in S'$, regardless of its input. Let $C_{\mathsf{seed.row}} := (C_1^a, C_2^a, \ldots, C_{d_S}^a, C_1^{pc}, C_2^{pc}, \ldots, C_{d_{S'}}^{pc})$ where $d_S := |S|$ and $d_{S'} := |S'|$, that is, $C_{\mathsf{seed.row}}$ is a circuit with $d_S + d_{S'}$ outputs and each of its outputs is a circuit C_ι^a or C_ι^{pc}.

Now we define the $\mathsf{Prod}\circ\mathsf{Sum}$ circuit C^{Prod}.

Circuit C^{Prod}

(Inputs) The input y has the form $y = y_{\mathsf{seed.row}} = (y_1, y_2, \ldots, y_\ell)$ and the input $\hat{\alpha}$ has the form $\hat{\alpha} = \hat{\alpha}_{\mathsf{seed.row}} = (\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_{d_S})$. The intended meanings are $y_i = C_{\mathsf{seed.row}}(z_i)$, and $\hat{\alpha}_i = \alpha_{\mathsf{irow}[i]}$.

For convenience, we will use the following labels to refer to bits of y, assuming the intended meaning above:

- For $j \in S^{\mathsf{proof}}$, $i \in [\ell]$, let $(y_i)_j := C^a_j(z_i) = C_{\mathsf{irow}[j]}(w_i)$;
- For $j \in S^{\mathsf{input}}$, $i \in [W_{\mathsf{input}}]$, let $(y_i)_j := C^a_j(z_i) = \hat{C}_{\tilde{x}_{\mathsf{irow}[j]}}(i) = \tilde{x}_{\mathsf{irow}[j],i}$;
- For $j \in S'$, let $(y_i)_{j+d_S} := C^{pc}_j(z_i) = pc^{\mathsf{row}}_j$.

(Linear sum gates) There are $\ell_{\mathsf{Sum}} := W_{\mathsf{proof}} \cdot d_S + 2d_{S'}$ linear sum gates and we identify $[\ell_{\mathsf{Sum}}]$ with the disjoint union of $[W_{\mathsf{proof}}] \times S$ and $S' \times \{0, 1\}$.

Let $i \in [W_{\mathsf{proof}}]$ and $j \in S$. If $\mathsf{itype}[j] = \mathsf{proof}$, then the (i, j)-th linear sum gate is

$$\mathsf{Sum}_{(i,j)}(y, \alpha) = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_j) \cdot (y_{\mathsf{idx}_k(\alpha_j, i)})_j.$$

It is easy to verify that

$$\mathsf{Sum}_{(i,j)}(y, \alpha) = (\mathsf{dec}_{\alpha_{\mathsf{irow}[j]}}(C_{\mathsf{irow}[j]}(w)))_i.$$

On the other hand, if $\mathsf{itype}[j] = \mathsf{input}$, then the (i, j)-th linear sum gate is $\mathsf{Sum}_{(i,j)}(y, \alpha) := (y_i)_j$. (If $i > W_{\mathsf{input}}$ then we simply set $\mathsf{Sum}_{(i,j)}(y, \alpha) = 0$ and this gate would not be used.)

Finally, for each $j \in S'$, we have two intermediate gates

$$\mathsf{Sum}_{(j,0)}(y, \alpha) = (y_1)_{j+d_S}, \quad \mathsf{Sum}_{(j,1)}(y, \alpha) = 1 - (y_1)_{j+d_S}.$$

Implementation of the linear sum layer: The linear sum has fan-in $A' := A \cdot d_S + 2$ and we identify $[A']$ with the disjoint union of $[A] \times S$ and $\{+, -\}$. Also, the length of y is $\ell_y := \ell \cdot (d_S + d_{S'})$, and we identify $[\ell_y]$ with $[\ell] \times (S \cup S')$. Let idx' and coeff' be the idx and coeff functions of the linear sum layer of C^{Prod}, then

(Function $\mathsf{idx}'_k(\alpha, i)$) Suppose $i = (i', j) \in [W_{\mathsf{proof}}] \times S$. If $\mathsf{itype}[j] = \mathsf{proof}$ and $k = (k', j')$ where $j = j'$, then we return $\mathsf{idx}'_k(\alpha, i) = (\mathsf{idx}_{k'}(\alpha_j, i'), j)$; if $\mathsf{itype}[j] = \mathsf{input}$ and $i' \in [W_{\mathsf{input}}]$ and $k = +$, then $\mathsf{idx}'_k(\alpha, i) = (i', j)$. Otherwise $\mathsf{idx}'_k(\alpha, i) = \mathsf{ZERO}$.

On the other hand, suppose $i = (j, b) \in S' \times \{0, 1\}$. If ($b = 0$ and $k = +$) or ($b = 1$ and $k = -$) then $\mathsf{idx}'_k(\alpha, i) = (1, j)$. If $b = 1$ and $k = +$ then $\mathsf{idx}'_k(\alpha, i) = \mathsf{ONE}$. Otherwise $\mathsf{idx}'_k(\alpha, i) = \mathsf{ZERO}$.

(Function $\mathsf{coeff}'_k(\alpha)$) If $k = +$ then $\mathsf{coeff}'_k(\alpha) = 1$; if $k = -$ then $\mathsf{coeff}'_k(\alpha) = -1$; otherwise, if $k = (k', j')$ then $\mathsf{coeff}'_k(\alpha) = \mathsf{coeff}_{k'}(\alpha_{j'})$.

The locality of $(\mathsf{idx}', \mathsf{coeff}')$ is still l. The coefficient sum becomes $d_S \cdot U + 2$.

(Output product gates) There are $2^{r_{\mathsf{col}}}$ product gates. For each $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output gate is

$$C^{\mathsf{Prod}}_{\mathsf{seed.col}}(y, \alpha) = \prod_{j \in S} \mathsf{Sum}_{(\mathsf{icol}[j], j)}(y, \alpha) \cdot \prod_{j \in S'} \mathsf{Sum}_{(j, pc^{\mathsf{col}}_j)}(y, \alpha).$$

To summarise, the parameters of the circuit C^{Prod} are as follows.

- The number of gates in each layer: $\ell_{\mathsf{Sum}} = W_{\mathsf{proof}} \cdot d_S + 2d_{S'}$, $\ell_{\mathsf{Prod}} = 2^{r_{\mathsf{col}}}$.
- The length of input y: $\ell_y = \ell(d_S + d_{S'})$;
- The fan-in of the top Prod gates: $d_S + d_{S'} \leq 2q$.
- The fan-in $A' := A \cdot d_S + 2$, coefficient sum $d_S \cdot U + 2$, and locality l of the linear sum layer.

Given the above construction, it is easy to see that Eq. (6) holds for every $\mathsf{seed.row}$ and $\mathsf{seed.col}$. We can thus see that

$$p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S') = \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}}\\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}}\left[C^{\mathsf{Prod}}_{\mathsf{seed.col}}(C_{\mathsf{seed.row}}(w), \alpha_{\mathsf{seed.row}})\right].$$

Since $d_S \le q, d_{S'} \le q$ and $\ell_{\mathscr{C}} \le d_S + d_{S'} \le 2q$, by Lemma 4.5, we can estimate $p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S')$ with additive error $\eta \cdot (qU+2)^{2q}$ in deterministic time

$$(A \cdot d_S + 2)^{2q} \cdot ((2q)^{2q} + 2^{r_{\mathsf{col}}}/N) \cdot (2^{2ql} + 2^{r_{\mathsf{row}}}/N) \cdot O(T^{\mathrm{alg}})$$

Analysis. First, the verification step takes

$$\begin{aligned}
&O((3A)^{4q} T^{\mathrm{alg}}) \cdot \left(2^{4ql + r_{\mathsf{shared}}} + T \log^{O(m)} T / 2^{2r_{\mathsf{col}}}\right)\\
\le\ &O((3A)^{4q}) \cdot (T \log^{O(m)} T)/(r_{\mathsf{col}})^{c_u \log(1/\varepsilon)}\\
\le\ &T(\log T)^{O(m) - c_u \log(1/\varepsilon)/2}
\end{aligned}$$

time, which is at most $T/(4\log^{c_{\mathsf{hard}}} T)$ if c_u is a large enough constant.

Then, the algorithm of Lemma 4.5 on C^{Prod} runs for every $\mathsf{seed.shared}, S, S'$, and in total takes time

$$\begin{aligned}
&(A \cdot d_S + 2)^{2q} \cdot ((2q)^{2q} + 2^{r_{\mathsf{col}}}/N) \cdot (2^{2ql} + 2^{r_{\mathsf{row}}}/N) \cdot O(T^{\mathrm{alg}}) \cdot 2^{2q} \cdot 2^{r_{\mathsf{shared}}}\\
\le\ &O(\log^{2q} \ell/\varepsilon^{4q}) \cdot O(2^{r_{\mathsf{col}}}/N) \cdot O(2^{r_{\mathsf{row}}}/N) \cdot O(N^2/\log^{c_u \log(1/\varepsilon)} N) \cdot 2^{r_{\mathsf{shared}}}\\
\le\ &2^r/\log^{\Omega(c_u)} \ell < T/(4\log^{c_{\mathsf{hard}}} T),
\end{aligned}$$

when c_u is sufficiently large. Therefore, the whole algorithm runs in $T/\log^{c_{\mathsf{hard}}} T$ time.

Besides, by the same argument as in the proof of Theorem 3.5, which we omit here, the algorithm estimates p_{acc} within additive error at most

$$\eta(4qU+8)^{2q} + 16^q \cdot 100^{-q} < 1/6,$$

thus successfully distinguishes between the case that $p_{\mathsf{acc}} > 5/6$ and that $p_{\mathsf{acc}} < 1/2$.

Description of M^{PCPP} We summarise the algorithm M^{PCPP}. On input x, we consider the smooth and rectangular PCPP for the language $L^{\mathsf{enc}} = \{\mathsf{Enc}(x) : x \in L^{\mathsf{hard}}\}$. (Recall that M^{PCPP} aims to reject every $x \notin L$ and accepts every $x \in L$ with easy witness.) We guess $(C_1, \ldots, C_{H_{\mathsf{proof}}})$ and $(\alpha_1, \ldots, \alpha_{H_{\mathsf{proof}}})$, which implicitly defines the PCPP proof matrices π^{Bool} and π^{Real}. Then we verify π^{Real} using Lemma 3.8 and reject immediately if π^{Real} did not pass the test. If π^{Real} passes the test (which means that it is "close" to a Boolean proof π^{Bool}), we use the algorithm described above to estimate p_{acc}. We accept x if and only if our estimation is above $2/3$.

The correctness of M^{PCPP} is easy to see (and is exactly the same as Claim 3.11):

Claim 4.7. *For every input x, if $x \notin L$ then M^{PCPP} rejects x; while if $x \in L$ and x has an easy witness then M^{PCPP} accepts x.*

The machine M^{PCPP} guesses $H_{\mathsf{proof}}(5s\log s + a) < n_{\mathsf{hard}}/10$ bits of nondeterminism (the number of size-s $\mathscr{C}$ circuits is at most $2^{5s\log s}$), and uses $\ell^{4q} < n_{\mathsf{hard}}/10$ bits of advice. Thus it computes a language in $\mathsf{NTIMEGUESS}_{\mathrm{RTM}}[T/\log^{c_{\mathsf{hard}}}(T), n_{\mathsf{hard}}/10]_{/(n_{\mathsf{hard}}/10)}$.

The $\mathsf{FP}^{\mathsf{NP}}$ algorithm for average-case hard partial truth table. Let $w_1, w_2, \ldots, w_\ell \in \{0,1\}^n$ be the input. We first construct the hard language L^{hard} and the algorithm M^{PCPP}. Since M^{PCPP} is a nondeterministic RAM algorithm that runs in $T/\log^{c_{\mathsf{hard}}}(T)$ time, uses at most $n_{\mathsf{hard}}/10$ nondeterministic bits and at most $n_{\mathsf{hard}}/10$ advice bits, it follows that there is an input $x_{\mathsf{hard}} \in \{0,1\}^{n_{\mathsf{hard}}}$ such that $M^{\mathsf{PCPP}}(x_{\mathsf{hard}}) \neq L^{\mathsf{hard}}(x_{\mathsf{hard}})$. Moreover, let α be the advice string fed to M^{PCPP}, i.e., the circuit C. We can find such an input x_{hard} by running $\mathcal{R}(1^{n_{\mathsf{hard}}}, M^{\mathsf{PCPP}}, \alpha)$, where $\mathcal{R}$ is the refuter guaranteed by Theorem 2.2. Thus, we can find x_{hard} in deterministic $\mathrm{poly}(T)$ time with an NP oracle.

It follows from Claim 4.7 that $x_{\mathsf{hard}} \in L^{\mathsf{hard}}$ but x_{hard} does not have an easy witness. Thus, we can use the NP oracle to find the lexicographically first PCPP proof matrix π such that

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

Then, there must exist a row π_i such that $\mathsf{Amp}(\pi_i)$ is $(1/2-\varepsilon)$-far from $C(w) = C(w_1)\circ\cdots\circ C(w_\ell)$ for any size-s $\mathscr{C}$ circuit C. To see this, suppose that for every i, there exists a size-s $\mathscr{C}$ circuit C_i such that $\mathsf{Amp}(\pi_i)$ is $(1/2-\varepsilon)$-close to $C_i(w)$. By Theorem 2.5, there is an advice α_i such that $\mathsf{dec}_{\alpha_i}(C_i(w))$ satisfies (1) for every $j \in [W_{\mathsf{proof}}]$, $(\mathsf{dec}_{\alpha_i}(C_i(w)))_j \in [0,1]$; (2) $\|\mathsf{dec}_{\alpha_i}(C_i(w)) - \pi_i\|_1 \leq \delta$. It follows that π is an easy witness for x_{hard}, a contradiction.

Finally, we use the NP oracle to find the first row π_i, such that $\mathsf{Amp}(\pi_i)$ is $(1/2-\varepsilon)$-far from $C(w)$ for any size-s $\mathscr{C}$ circuit C, and output $\mathsf{Amp}(\pi_i)$. The overall procedure takes deterministic $\mathrm{poly}(T) \leq \mathrm{poly}(\ell)$ time with an NP oracle. □

5 Unconditional Algorithms for Range Avoidance, Remote Point, and Hard Partial Truth Tables

In this section, we apply the frameworks in Section 3 and Section 4 to obtain *unconditional* results for XOR-Remote-Point, ACC^0-Remote-Point, and ACC^0-Partial-AvgHard.

5.1 Algorithms for Satisfying Pairs

In this section, we present some algorithms for $\mathscr{C}$-Satisfying-Pairs for various circuit classes $\mathscr{C}$ that run in non-trivial time.

The algorithms in this section require the following algorithm for the batch evaluation of low-degree polynomials via fast rectangular matrix multiplication. This algorithm has been extensively used in previous works on the polynomial method and circuit complexity (see, e.g., [Wil14a, Wil18a]). We provide a proof sketch for completeness.

Theorem 5.1. *Let $x_1, x_2, \ldots, x_N \in \{0,1\}^n$ be N input strings, and $p_1, p_2, \ldots, p_N : \{0,1\}^n \to \mathbb{N}$ be N integer polynomials of degree at most d. Suppose that $n^{20d} \leq N$. Then there is a deterministic algorithm running in $\tilde{O}(N^2)$ time that outputs the table of $p_j(x_i)$ for every $i, j \in [N]$.*

Theorem 5.2 ([Cop82]; see also [Wil18a, Appendix C])**.** *There is a (deterministic) algorithm for multiplying an $N \times N^{0.1}$ matrix and an $N^{0.1} \times N$ matrix using $\tilde{O}(N^2)$ arithmetic operations.*

Proof of Theorem 5.1. There are $m := \sum_{i=0}^{d} \binom{n}{i} \leq (en/d)^d \leq N^{0.1}$ monomials of degree at most d. We number these monomials from 1 to m. Let S_j denote the set of indices in the j-th monomial. That is, the j-th monomial is $\prod_{k \in S_j} x_k$.

We construct two matrices $M_1 \in \mathbb{Z}^{N\times m}$ and $M_2 \in \mathbb{Z}^{m\times N}$. For each $i \in [N]$ and $j \in [m]$, $M_1[i,j]$ is the evaluation of the j-th monomial on input x_i. (That is, $M_1[i,j] = \prod_{k\in S_j}(x_i)_k$.) For each $j \in [m]$ and $k \in [N]$, $M_2[j,k]$ is the coefficient of the j-th monomial in p_k.

Let $M := M_1 \cdot M_2$. It follows that for every $i, j \in [N]$, $M[i,j] = p_j(x_i)$. Since $m \le N^{0.1}$, we can compute M in $\tilde{O}(N^2)$ time using Theorem 5.2. □

5.1.1 An Algorithm for #XOR-Satisfying-Pairs

We start by demonstrating a non-trivial algorithm for #XOR-Satisfying-Pairs, i.e., the exact counting version of Satisfying-Pairs problem where each circuit consists of a single XOR gate (with unbounded fan-in). We observe that #XOR-Satisfying-Pairs is essentially $\mathbb{F}_2$-#OV (i.e., the counting version of Orthogonal Vectors problem over $\mathbb{F}_2$), which has been studied in the literature before (see, e.g., [CW21, AC19]).

In more details, an unbounded fan-in XOR gate over n input bits can be represented by a vector $y \in \{0,1\}^n$ and a bit $b \in \{0,1\}$ such that on input $x \in \{0,1\}^n$, the gate outputs 1 if $\langle x, y\rangle = b$ and outputs 0 otherwise. Suppose that we have N inputs $x_1, x_2, \ldots, x_N \in \{0,1\}^n$ and N XOR gates $(y_1, b_1), \ldots, (y_N, b_N)$. Let $x'_i \in \{0,1\}^{n+1}$ be the vector that is the concatenation of x_i and the bit 1, and let $y'_j \in \{0,1\}^{n+1}$ be the vector that is the concatenation of y_j and b_j. Then the gate (y_j, b_j) accepts the input x_i if and only if $\langle x'_i, y'_j\rangle = 0$. Hence we can reduce #XOR-Satisfying-Pairs to the counting version of $\mathbb{F}_2$-OV, which has a non-trivial algorithm [AC19, CW21].

Theorem 5.3 ([AC19, Theorem 2.9])**.** *Let $n \le N^{o(1)}$. There is a deterministic algorithm running in $N^{2-\Omega(1/\log(n/\log N))}$ time that given $2N$ length-n vectors $x_1, x_2, \ldots, x_N, y_1, y_2, \ldots, y_N \in \mathbb{F}_2^n$, outputs the number of pairs (i,j) such that $\langle x_i, y_i\rangle = 0$.*

5.1.2 An Algorithm for #ACC^0-Satisfying-Pairs

We present a non-trivial algorithm for #ACC^0-Satisfying-Pairs, generalising the algorithm for XOR-circuits (since XOR $\subseteq$ AC$^0[2] \subseteq$ ACC^0). This algorithm utilises a quasi-polynomial simulation of SYM $\circ$ ACC^0 circuits with SYM $\circ$ AND circuits.

Theorem 5.4 (From SYM $\circ$ ACC^0 to SYM $\circ$ AND [BT94, AG91, Wil18c])**.** *Let m, ℓ be any constants, there exists an integer c' such that every SYM $\circ$ AC$^0_\ell[m]$ circuit of size s can be simulated by a SYM $\circ$ AND circuit of $2^{(\log s)^{c'}}$ size. Moreover, the AND gates of the final circuit have only $(\log s)^{c'}$ fan-in, the final circuit can be constructed from the original one in $2^{O((\log s)^{c'})}$ time, and the final symmetric function at the output can be computed in $2^{O((\log s)^{c'})}$ time.*

Combining Theorem 5.4 with Theorem 5.1, we can derive the #ACC^0-Satisfying-Pairs algorithm in non-trivial time as follows.

Theorem 1.16. *For every constants m, ℓ, c, there is a constant $\varepsilon \in (0,1)$ such that the following holds. Let $n := 2^{\log^\varepsilon N}$ and $s := 2^{\log^c n}$. There is a deterministic algorithm running in $\tilde{O}((N/n)^2)$ time that given N strings $x_1, x_2, \ldots, x_N \in \{0,1\}^n$ and N AC$^0_\ell[m]$ circuits $C_1, C_2, \ldots, C_N : \{0,1\}^n \to \{0,1\}$ of size s, outputs the number of pairs $(i,j) \in [N] \times [N]$ such that $C_i(x_j) = 1$.*

Proof. Let ε be a constant to be determined. We divide $C_1, C_2, \ldots, C_N$ into N/n groups where each group has size n. Let C_{ij} denote the j-th circuit in the i-th group. We also partition the inputs $x_1, x_2, \ldots, x_N$ into N/n groups of size n and define x_{ij} similarly. Let $X_i := x_{i1} \circ x_{i2} \circ \cdots \circ x_{in}$.

For each group i, we can construct $g := \lceil 2\log n \rceil$ $\mathsf{SYM} \circ \mathsf{AC}^0_\ell[m]$ circuits $D_{i1}, D_{i2}, \ldots, D_{ig} : \{0,1\}^{n^2} \to \{0,1\}$, each of size $s' := O(n^2 \cdot s)$, such that for any group j, we have:

$$\sum_{i'=1}^{n}\sum_{j'=1}^{n} C_{ii'}(x_{jj'}) = \sum_{k=0}^{g} 2^k D_{ik}(X_j).$$

That is, $D_{ik}(X_j)$ computes the k-th bit of the number of satisfying pairs between the i-th group of circuits and the j-th group of inputs.

Let c' be the constant in Theorem 5.4 depending on ℓ and m. We can transform each $\mathsf{SYM} \circ \mathsf{AC}^0_\ell[m]$ circuit D_{ij} into a $\mathsf{SYM} \circ \mathsf{AND}$ circuit D'_{ij} of size $2^{(\log s')^{c'}}$ such that each AND gate has fan-in at most $d := (\log s')^{c'}$. We can write each $D'_{ij}(x)$ as $f_{ij}(p_{ij}(x))$, where $p_{ij}(x) : \{0,1\}^{n^2} \to \{0,1,\ldots,2^{(\log s')^{c'}}\}$ is a polynomial of degree at most d that only outputs integers upper bounded by $2^{(\log s')^{c'}}$ on Boolean inputs, and f_{ij} is some function that can be evaluated in $2^{O((\log s')^{c'})}$ time. We can construct the polynomials p_{ij} and (the truth tables of) the functions f_{ij} in $(N/n)g2^{O((\log s')^{c'})}$ time. Let $\varepsilon := 1/(10cc')$ (and recall $n = 2^{\log^\varepsilon N}$ and $s = 2^{\log^c n}$), this time bound is at most $(N/n)^2$.

Then, for each $k = 1, 2, \ldots, g$, since $(n^2)^{20d} \le N/n$, we can compute the table of $p_{ik}(X_j)$ for every $i, j \in [N/n]$ in $\tilde{O}((N/n)^2)$ time by invoking Theorem 5.1. In fact, by checking the truth-tables of f_{ij}, we actually get the table for $D'_{ik}(X_j) = D_{ik}(X_j)$. Finally it follows that:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} C_i(x_j) = \sum_{i=1}^{N/n}\sum_{j=1}^{N/n}\sum_{i'=1}^{n}\sum_{j'=1}^{n} C_{ii'}(x_{jj'}) = \sum_{i=1}^{N/n}\sum_{j=1}^{N/n}\sum_{k=0}^{g} 2^k D_{ik}(X_j).$$

The total run-time is bounded by $2(N/n)^2 + g\tilde{O}((N/n)^2) = \tilde{O}((N/n)^2)$. □

5.2 An $\mathsf{FP}^{\mathsf{NP}}$ Algorithm for XOR-Remote-Point

As a warm-up, we show an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for XOR-Remote-Point. This result is subsumed by the algorithm for ACC-Remote-Point (Theorem 1.17) as well as previous results on rigid matrices [AC19, BHPT20, CLW20, CL21, HV21], but we still mention it for several reasons:

1. This is an important special case for Remote-Point.
2. This result only requires the algorithm for $\#\mathbb{F}_2$-OV, and is a nice illustration that many special cases of Satisfying-Pairs problems are widely studied by algorithm designers.

Theorem 1.15 (XOR-Remote-Point $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \ge 1$ such that the following holds. Let $\varepsilon := \varepsilon(n) \ge 2n^{-c_u}$ be the error parameter and $\ell := \ell(n) \ge 2^{\log^{c_u+5} n}$ be the stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where each output bit of C is computed by an XOR gate, and outputs a string y that is $(1/2-\varepsilon)$-far from $\mathsf{Range}(C)$.*

Proof. Let c_u be the constant in Theorem 3.5. In order to make XOR a typical circuit class (closed under negation and projection), we here allow XOR using bias, denoted by $\widetilde{\mathsf{XOR}}$. That is, an $\widetilde{\mathsf{XOR}}$ gate g can be represented by parameters $w \in \{0,1\}^n$ and $b \in \{0,1\}$, such that for any $x \in \{0,1\}^n$, we have $g(x) := \langle w, x\rangle \oplus b$. To obtain an $\mathsf{FP}^{\mathsf{NP}}$ algorithm of XOR-Remote-Point by Theorem 3.5, we need to design an efficient algorithm for $\mathsf{AND}_{c_u} \circ \widetilde{\mathsf{XOR}}$-Satisfying-Pairs.

Since an $\widetilde{\mathsf{XOR}}$ gate can be regarded as an XOR gate with (or without) a NOT gate on its output wire, it follows that $\mathsf{AND}_{c_u} \circ \widetilde{\mathsf{XOR}} \subseteq \mathsf{NC}^0 \circ \mathsf{XOR}$. Furthermore, by Theorem 2.17, It suffices to solve

$\#\mathsf{XOR}_{c_u} \circ \mathsf{XOR}$-Satisfying-Pairs with roughly the same parameters. As $\mathsf{XOR}_{c_u} \circ \mathsf{XOR} \subseteq \mathsf{XOR}$, we only need to consider $\#\mathsf{XOR}$-Satisfying-Pairs.

Let $d := \max\{c_u, 2.02\}$ be a constant, we apply Theorem 5.3 with $N := N(n) = 2^{\log^d n}$, $n_{\mathsf{sat}} := 2^{\log^{0.499} N} \geq 2^{\log^{1.001} n} \geq \mathrm{poly}(n)$ as parameters. (That is, we pad $n_{\mathsf{sat}} - n$ bits to the inputs of the Satisfying-Pairs instance.) It gives an algorithm for $\#\mathsf{XOR}$-Satisfying-Pairs (and therefore also for $\mathsf{AND}_{c_u} \circ \mathsf{XOR}$) with running time $T = N^{2-\Omega(1/\log(n_{\mathsf{sat}}/\log N))}$. We can check the technical requirement for T:

$$\begin{aligned} T &= N^{2-\Omega(1/\log(n_{\mathsf{sat}}/\log N))} \leq N^2 2^{-\Omega(\log N/\log n_{\mathsf{sat}})} \leq N^2 2^{-\log^{0.5001} N} \\ &\leq N^2 2^{-\log^{1.01} n} \leq N^2 2^{-c_u^2 \log n \log\log N} \leq N^2/P^{c_u}, \end{aligned}$$

where $P := (\log N)^{\log(1/\varepsilon)}$. Other technical requirements for Theorem 3.5 can be easily verified, so we can invoke it and get an $\mathsf{FP}^{\mathsf{NP}}$ Remote Point algorithm for XOR gates with error $\varepsilon'(n) > n^{-c_u}$ and stretch $\ell'(n) = N^{c_u \log(1/\varepsilon)}$. Finally, we can check $\ell(n) > \ell'(n+1)$ and $\varepsilon(n) > 2\varepsilon'(n+1)$, then get a desired $\mathsf{FP}^{\mathsf{NP}}$ algorithm for original parameters by Lemma 2.15. □

5.3 Remote Point for ACC^0

Theorem 1.17 (ACC^0-Remote-Point $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \geq 1$ such that for every constant $d, m \geq 1$, there is a constant $c_{\mathsf{str}} := c_{\mathsf{str}}(d,m) \geq 1$, such that the following holds.*

Let $n < s(n) \leq 2^{n^{o(1)}}$ be a size parameter, $\varepsilon := \varepsilon(n) \geq 2n^{-c_u}$ be an error parameter and $\ell := \ell(n) \geq 2^{\log^{c_{\mathsf{str}}} s}$ be a stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that takes as input a circuit $C : \{0,1\}^n \to \{0,1\}^\ell$, where each output bit of C is computed by an $\mathsf{AC}^0_d[m]$ circuit of size s, and outputs a string y that is $(1/2-\varepsilon)$-far from $\mathsf{Range}(C)$.

Proof. Let c_u be the constant from Theorem 3.5, and c_{str} be a constant to be determined later. We then set parameters for invoking Theorem 3.5.

We set $n_{\mathsf{sat}} := \max\{2^{\log^{c_u+2} n}, 2^{\log^{c_u+1} s}\}$. Then we can invoke Theorem 1.16 with input length n_{sat} and size parameter also n_{sat} to get a $\#\mathsf{AC}^0_{d+1}[m]$-Satisfying-Pairs algorithm for N circuits and N inputs, where $N := N(n) = 2^{\log^{1/\varepsilon_{\mathsf{sat}}} n_{\mathsf{sat}}}$ for some constant $\varepsilon_{\mathsf{sat}} \in (0,1)$. This algorithm runs in time $T = \tilde{O}((N/n_{\mathsf{sat}})^2)$.

Set $c_{\mathsf{str}} := (c_u+2)/\varepsilon_{\mathsf{sat}} + 3$. Let $\varepsilon' := \varepsilon'(n) \geq n^{-c_u}$ be the error parameter and $\ell' := \ell'(n) = N^{c_u \log(1/\varepsilon')}$ be the stretch, then we can check these parameters satisfy the requirements of Theorem 3.5 as follows.

$$\begin{aligned} &2^{\log^{c_u} n} \leq N \leq 2^{n^{0.99}} \\ &\ell'(n) = N^{c_u \log(1/\varepsilon')} \geq c_u s \\ &T \leq N^2/2^{\log^{c_u+2} n} \leq N^2/n^{c_u^2 \log^{c_u} n} \leq N^2/(\log N)^{c_u^2 \log^{c_u} n} \leq N^2/P^{c_u} \end{aligned}$$

(That is, we use the aforementioned algorithm to solve Satisfying-Pairs with N circuits of size s and N inputs of length n by padding $n_{\mathsf{sat}} - n$ dummy bits to each input, and then apply Theorem 3.5). By Theorem 3.5, we get an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for ACC^0-Remote-Point with error $\varepsilon'(n) > n^{-c_u}$, and stretch $\ell'(n) = N^{c_u \log(1/\varepsilon)}$. We can check that both $\ell(n) > \ell'(n+1)$ and $\varepsilon(n) > 2\varepsilon'(n+1)$ hold, so we can invoke Lemma 2.15 and get a desired $\mathsf{FP}^{\mathsf{NP}}$ algorithm for remote point with the original parameters. □

We can easily recover the state-of-the-art almost-everywhere average-case circuit lower bounds against ACC^0 [CLW20] by giving the truth table generator as the input.

Corollary 5.5. *For every constants $d, m \geq 1$, there is an $\varepsilon > 0$ and a language $L \in \mathsf{E}^{\mathsf{NP}}$ such that L_n cannot be $(1/2 + 2^{-n^{\varepsilon}})$-approximated by $\mathsf{AC}^0_d[m]$ circuits of size $2^{n^{\varepsilon}}$, for all sufficiently large n.*

Proof Sketch. Let $\mathsf{TT}_s : \{0,1\}^{O(s \log s)} \to \{0,1\}^{2^n}$ be the truth table generator of $\mathsf{AC}^0_d[m]$ circuits, where $s = 2^{n^{\varepsilon}}$ for some constant ε to be determined later. Each output bit of TT_s is computable by an $\mathsf{AC}^0_{d'}[m]$ circuit of size $s' = \mathrm{poly}(s)$ for some $d' = O(1)$.

For clarity we define $n_{\mathsf{tt}} = O(s \log s)$ to be the input length of TT_s, $s_{\mathsf{tt}}(n_{\mathsf{tt}}) := s'$ and $d_{\mathsf{tt}} := d'$ to be the size and depth of TT_s, respectively. Let c_u and $c_{\mathsf{str}} := c_{\mathsf{str}}(d_{\mathsf{tt}}, m)$ be the constants in Theorem 1.17. Then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm A_{hard} that takes as input a circuit $C : \{0,1\}^{n_{\mathsf{tt}}} \to \{0,1\}^{\ell_{\mathsf{tt}}}$ and outputs a string y that is $(1/2 - \varepsilon_{\mathsf{tt}})$-far from $\mathsf{Range}(C)$, where $\ell_{\mathsf{tt}} \geq 2^{\log^{c_{\mathsf{str}}} s_{\mathsf{tt}}}$ and $\varepsilon_{\mathsf{tt}} := 2n_{\mathsf{tt}}^{-c_u}$. By choosing ε to be a sufficiently small constant, we can make

$$2^n \geq 2^{\log^{c_{\mathsf{str}}} s_{\mathsf{tt}}} \qquad \text{and} \qquad 2^{-n^{\varepsilon}} > \varepsilon_{\mathsf{tt}}.$$

We then fix the input of the $\mathsf{FP}^{\mathsf{NP}}$ algorithm A_{hard} above to be TT_s to obtain an $\mathsf{FP}^{\mathsf{NP}}$ algorithm A that takes 1^{2^n} as input and produces a truth table of length 2^n that cannot be $(1/2 + 2^{-n^{\varepsilon}})$-approximated by any size-s circuits. The required hard language is then defined as

$$L := \left\{ x \in \{0,1\}^n : n \in \mathbb{N}, \mathsf{tt} \leftarrow A(1^{2^n}) \in \{0,1\}^{2^n}, \mathsf{tt}_x = 1 \right\}. \qquad \square$$

5.4 Hard Partial Truth Tables for ACC^0

Theorem 1.18 (ACC^0-Partial-AvgHard $\in \mathsf{FP}^{\mathsf{NP}}$)**.** *There is a constant $c_u \geq 1$ such that for every constants $d, m \geq 1$, there is a constant $c_{\mathsf{str}} := c_{\mathsf{str}}(d, m) \geq 1$, such that the following holds.*

Let $n < s(n) \leq 2^{n^{o(1)}}$ be a size parameter, $\varepsilon := \varepsilon(n) \geq 2n^{-c_u}$ be an error parameter and $\ell := \ell(n) \geq 2^{\log^{c_{\mathsf{str}}} s}$ be a stretch function, then there is an $\mathsf{FP}^{\mathsf{NP}}$ algorithm that given inputs $x_1, \ldots, x_\ell \in \{0,1\}^n$, it outputs a string $y \in \{0,1\}^\ell$ such that for any $s(n)$-size $\mathsf{AC}^0_d[m]$ circuit C, y is $(1/2 - \varepsilon)$-far from $C(x_1) \circ \cdots \circ C(x_\ell)$.

Proof Sketch. The proof is similar to Theorem 1.17, so we only sketch the proof.

Let c_u be the constant from Theorem 4.3, and then we set values[19] of $c_{\mathsf{str}}, n_{\mathsf{sat}}, N, \varepsilon_{\mathsf{sat}}, T, \varepsilon'(n)$ and $\ell'(n)$ in the same way as proof Theorem 1.17.

Since parameter constraints of Theorem 4.3 are similar to those of Theorem 3.5, These parameter settings can be used to invoke Theorem 4.3 and get an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $O(s(n))$-size $\mathsf{AC}_{d+O(1)}[m]$-Partial-AvgHard with stretch $\ell'(n)$ and error $\varepsilon'(n)$. We can check that both $\ell(n) > \ell'(n+1)/2$ and $\varepsilon(n) > 2\varepsilon'(n+1)$ hold, so it is valid to invoke Lemma 2.16 and get a desired $\mathsf{FP}^{\mathsf{NP}}$ algorithm for average-case hard partial truth tables with the original parameters.

Alternatively, we can reduce ACC^0-Partial-AvgHard to ACC^0-Remote-Point (see Section 1.3) and simply apply Theorem 1.17, since the evaluation of ACC^0 circuits can be implemented in ACC^0. $\square$

As a consequence, we show (following the observation in [AS10]) that there is no efficient mapping reduction from E^{NP} to any language decidable by small-size non-uniform ACC^0 circuits.

Corollary 1.20. *Let $d, m \in \mathbb{N}$ be constants, $\mathsf{AC}^0_d[m]$ denote the class of languages computable by a* non-uniform *family of polynomial-size $\mathsf{AC}^0_d[m]$ circuits. Then, there is a language $L^{\mathsf{hard}} \in \mathsf{E}^{\mathsf{NP}}$ that does not have polynomial-time mapping reductions to any language in $\mathsf{AC}^0_d[m]$.*

[19] In order to invoke Lemma 2.16, we actually use $s'(n) = O(s(n))$ as size function and $d' := d + O(1)$ as depth. These are rather minor changes, so we can still use the same parameter settings strategy.

Proof. Our E^{NP} language L^{hard} receives two inputs: a Turing machine R and a string y. Here, the lengths of $\langle R\rangle$ (the encoding of R) and y are $\lceil n/2\rceil$ and $n' := \lfloor n/2\rfloor$ respectively, thus L^{hard} receives n-bit strings as inputs. The machine R is interpreted as a reduction that runs in $T(n) := n^{\log n}$ time (which we diagonalise against).

We run R on all inputs of the form $(\langle R\rangle, x')$, where $|x'| = n'$. Let $x_1, x_2, \ldots, x_{2^{n'}}$ be an enumeration of length-n' strings, and $z_i := R(\langle R\rangle, x_i)$ be a string of length at most $T(n)$. Note that the strings z_i may not be of the same length, but the length of each z_i is at most $T(n)$. By an averaging argument, there is an $\ell \le T(n)$ such that there are at least $2^{n'}/T(n) \ge 2^{n^{0.99}}$ strings z_i with length exactly ℓ. Let N be the number of strings z_i with length exactly ℓ and denote these strings to be $z_{i_1}, z_{i_2}, \ldots, z_{i_N}$. We can check the technical constraints and invoke Theorem 1.18 to get an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for solving the $\mathsf{AC}^0_d[m]$-Partial-Hard problem on inputs $z_{i_1}, z_{i_2}, \ldots, z_{i_N}$. We obtain a sequence of bits $y_{i_1}, y_{i_2}, \ldots, y_{i_N} \in \{0,1\}$ such that for every size-$\ell^{\log \ell}$ $\mathsf{AC}^0_d[m]$ circuit C, there is some $j \in [N]$ such that $C(z_{i_j}) \neq y_{i_j}$. This can be done in deterministic $2^{O(n)}$ time with an NP oracle. Finally, we define L^{hard} as follows: suppose x is the i-th string of length n' (i.e., $x = x_i$), then $x \in L^{\mathsf{hard}}$ if and only if $|z_i| = \ell$ and $y_i = 1$.

Clearly, L^{hard} runs in deterministic $2^{O(n)}$ time with an NP oracle. We still need to show that for every language $L \in \mathsf{AC}^0_d[m]$, there is no polynomial time reduction from L^{hard} to L. Suppose, for the sake of contradiction, that there is a polynomial-time reduction R from L^{hard} to L. Let n be a sufficiently large number such that $n/2 > \langle R\rangle$ and $T(n) = n^{\log n}$ is larger than the running time of R. Consider running R on inputs of the form $(\langle R\rangle, x)$ where $|x| = \lfloor n/2\rfloor$. Let x_i, y_i, z_i, ℓ, and N be defined as above, and C be an $\mathsf{AC}^0_d[m]$ circuit that decides L on input length ℓ. Since the size of C is at most $\mathrm{poly}(\ell) \le \ell^{\log \ell}$, there is some $j \le N$ such that $C(z_{i_j}) \neq y_{i_j}$. In other words,

$$\exists i \in N, C(R(\langle R\rangle, x_i)) \neq L^{\mathsf{hard}}(\langle R\rangle, x_i).$$

It follows that R is not a correct reduction from L^{hard} to L. □

6 Construction of Smooth and Rectangular PCPP

Recall that a PCPP verifier is *smooth* if every bit of the proof is equally likely to be queried, i.e., the distribution of a random query position over a random seed is uniformly random. We do not impose any smoothness requirement on the input oracle.

In this section, we construct a smooth and rectangular PCP of proximity. There is essentially nothing new in our construction: With a careful inspection of their techniques, we can combine [RSW22] (a rectangular PCP of proximity) and [BHPT20] (a smooth and rectangular PCP) to obtain our PCPPs. This is not a coincidence as [BHPT20, RSW22] are both variants of [BGH+06]. Nevertheless, we present an (almost) self-contained proof of the construction for the convenience of the reader, as there are several components and many parameters in [BHPT20, RSW22]. As this section is quite technical, we give a brief overview of the construction.

Overview of Section 6

- Instead of constructing a smooth PCPP verifier directly, we will construct a rectangular PCPP verifier with the *rectangular neighbour listing* (RNL) property, following [BHPT20]. In Section 6.1, we will give the definition of RNL property and verify that the transformation from rectangular PCP verifiers with RNL property to smooth and rectangular PCP verifiers in [BHPT20] also works for PCPP verifiers.
- In Section 6.2, we show that the PCPP verifier in [BGH+06] is a robust and rectangular PCPP

verifier with RNL property. This combines the observations in [BHPT20] (for constructing robust and rectangular PCP verifiers with RNL property) and [RSW22] (for constructing robust and rectangular PCPP verifiers).

- In Section 6.3, we prove a *composition theorem* that generalizes the counterparts in [BHPT20, RSW22]: we can compose a robust and rectangular PCPP verifier for L (called the outer PCPP verifier) and a PCPP verifier for a variant of the circuit-evaluation problem (called the inner PCPP verifier) to obtain a rectangular PCPP verifier for L whose query complexity is at most the query complexity of the inner PCPP verifier. Moreover, the composed PCPP verifier has RNL property if the outer PCPP verifier has RNL property. Due to technical limitations, this rectangular PCPP verifier will also take some ROP parity-check bits (see Definition 2.9).
- By composing the robust PCPP verifier in Section 6.2 with a PCPP with query complexity $O(1)$, we can construct a rectangular PCPP verifier with query complexity $O(1)$ and RNL property. Moreover, we can transform it into a smooth and rectangular PCPP verifier with query complexity $O(1)$. However, the soundness error of this PCPP verifier could be a constant that is very close to 1. In Section 6.4, we show that the soundness error can be reduced to an arbitrarily small constant with an $O(1)$ blow-up to the query complexity, using an expander-walk-based technique developed in [RSW22, Section 7.1.3].
- We wrap all these components up and set the parameters in Section 6.5.

6.1 Rectangular Neighbour Listing and Smoothness

Definition 6.1. Let V be a rectangular PCPP verifier for some language L with randomness complexity r and query complexity q. A *configuration* is defined as a pair $(\mathsf{seed}, k) \in \{0,1\}^r \times [q]$. It is said to be a proof (resp. input) configuration if the verifier with randomness seed will query the proof (resp. input) oracle on the k-th query.

Two configurations (seed_1, k_1) and (seed_2, k_2) are said to be neighbours if the verifier will access the same bit of the same oracle with randomness seed_1 on the k_1-th query, or with randomness seed_2 on the k_2-th query.

Definition 6.2 (Rectangular Neighbour Listing)**.** Let L be a language and V be a rectangular PCPP verifier for L with row randomness complexity r_{row}, column randomness complexity r_{col}, and shared randomness complexity r_{shared}. We say V has $t_{\mathsf{RNL}}(n)$*-time rectangular neighbour listing* property if there are two $t_{\mathsf{RNL}}(n)$-time algorithms A_{row} and A_{col} such that the following conditions hold:

1. The shared randomness $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ consists of $\mathsf{seed.shared.row} \in \{0,1\}^{r_{\mathsf{shared}}/2}$ and $\mathsf{seed.shared.col} \in \{0,1\}^{r_{\mathsf{shared}}/2}$, i.e., $\mathsf{seed.shared} = (\mathsf{seed.shared.row}, \mathsf{seed.shared.col})$.
2. Let $(\mathsf{seed}, k) = (\mathsf{seed.row}, \mathsf{seed.col}, \mathsf{seed.shared}, k)$ be a configuration, where $\mathsf{seed.shared} = (\mathsf{seed.shared.row}, \mathsf{seed.shared.col})$. The algorithms A_{row} and A_{col} can list all the neighbours of (seed, k) in a "rectangular and synchronized" fashion:
   - Given the row-part randomness $(\mathsf{seed.row}, \mathsf{seed.shared}, k)$ as input, A_{row} will output an ordered list $\mathsf{NList}_{\mathsf{row}}(\mathsf{seed}, k) := \{(\mathsf{seed}_i.\mathsf{row}, \mathsf{seed}_i.\mathsf{shared.row}, k_i.\mathsf{row})\}_{i \in [\ell_{\mathsf{row}}]}$ and an index $\mathsf{self}_{\mathsf{row}}(\mathsf{seed}, k) \in [\ell_{\mathsf{row}}]$
   - Given the column part randomness $(\mathsf{seed.col}, \mathsf{seed.shared}, k)$ as input, A_{col} will output an ordered list $\mathsf{NList}_{\mathsf{col}}(\mathsf{seed}, k) := \{(\mathsf{seed}_i.\mathsf{col}, \mathsf{seed}_i.\mathsf{shared.col}, k_i.\mathsf{col})\}_{i \in [\ell_{\mathsf{col}}]}$ and an index $\mathsf{self}_{\mathsf{col}}(\mathsf{seed}, k) \in [\ell_{\mathsf{col}}]$.

Verifier	V^{old}	V^{new}
Soundness error	s	$s + \mu$
Proximity parameter	δ	δ
Row randomness	r_{row}	r_{row}
Column randomness	r_{col}	r_{col}
Shared randomness	r_{shared}	r_{shared}
Proof matrix height	H_{proof}	$q \cdot 2^{r_{\mathsf{row}}+r_{\mathsf{shared}}/2}$
Proof matrix width	W_{proof}	$2^{r_{\mathsf{col}}+r_{\mathsf{shared}}/2}$
Query complexity	q	$\mathrm{poly}(q/\mu)$
Parity check complexity	p	p
Decision complexity	d	$\mathrm{poly}(d, q/\mu, t_{\mathsf{RNL}}(n))$

Table 3: The parameters of the "smoothened" PCPP V^{new}.

- It holds that $\ell_{\mathsf{row}} = \ell_{\mathsf{col}}$ and $k_i.\mathsf{row} = k_i.\mathsf{col}$ for every $i \in [\ell_{\mathsf{row}}]$. Let $\ell := \ell_{\mathsf{row}}$ and $k_i := k_i.\mathsf{row}$ for every $i \in [\ell]$. Then the "zipped" list of $\mathsf{NList}_{\mathsf{row}}$ and $\mathsf{NList}_{\mathsf{col}}$

$$\mathsf{NList}(\mathsf{seed}, k) := \Big\{(\mathsf{seed}_i.\mathsf{row}, \mathsf{seed}_i.\mathsf{col}, \mathsf{seed}_i.\mathsf{shared.row}, \mathsf{seed}_i.\mathsf{shared.col}, k_i)\Big\}_{i\in[\ell]}$$

  is a list of all the neighbours of (seed, k). Moreover, $\mathsf{self}_{\mathsf{row}}(\mathsf{seed}, k) = \mathsf{self}_{\mathsf{col}}(\mathsf{seed}, k)$ and the $\mathsf{self}_{\mathsf{row}}(\mathsf{seed}, k)$-th entry of $\mathsf{NList}(\mathsf{seed}, k)$ is the configuration (seed, k) itself.
- For every pair of neighbours (seed_1, k_1) and (seed_2, k_2), $\mathsf{NList}(\mathsf{seed}_1, k_1) = \mathsf{NList}(\mathsf{seed}_2, k_2)$ (i.e., these two ordered lists are the same).

By slightly generalizing the technique of [BHPT20], we can smoothen a rectangular PCPP with the rectangular neighbour listing property.

Theorem 6.3. *Suppose that L has a rectangular PCPP verifier V^{old} with ROP and $t_{\mathsf{RNL}}(n)$-time RNL property, where the parameters are specified in Table 3. Then for every $\mu \in (0, 1)$, L has a smooth and rectangular PCPP verifier V^{new} with ROP and the parameters specified in Table 3.*

Proof. Let $\Pi^{\mathsf{old}} : \{0,1\}^{\ell} \to \{0,1\}$ be the proof oracle of V^{old}. Assume that $V^{\mathsf{old}}(\mathsf{seed}, i)$ outputs the index of the i-th query of V^{old} given the randomness $\mathsf{seed} \in \{0,1\}^r$. The "smoothened" verifier V^{new} expects the proof $\Pi^{\mathsf{new}} : \{0,1\}^{2^r} \times [q] \to \{0,1\}$ to be

$$\Pi^{\mathsf{new}}(\mathsf{seed}, i) := \Pi^{\mathsf{old}}[V^{\mathsf{old}}(\mathsf{seed}, i)].$$

Concretely, V^{new} works as follows: It firstly checks that Π^{new} is (close to being) defined as above, i.e., there is a proof matrix Π^{old} such that $\Delta(\Pi^{\mathsf{new}}(\mathsf{seed}, i), \Pi^{\mathsf{old}}[V^{\mathsf{old}}(\mathsf{seed}, i)])$ is sufficiently small; then it runs V^{old} using Π^{new} as the proof oracle, i.e., the verifier randomly chooses a $\mathsf{seed} \in \{0,1\}^r$, queries $\Pi^{\mathsf{new}}(\mathsf{seed}, 1), \Pi^{\mathsf{new}}(\mathsf{seed}, 2), \ldots, \Pi^{\mathsf{new}}(\mathsf{seed}, q)$, and decides whether to accept using the decision predicate of V^{old}. In fact, as in [BHPT20, Section 4.1], the first step can be combined into the second step: we only need to check the consistency of Π^{new} on the fly during the simulation of V^{old}.

The verifier V^{new}. For $\alpha \in (0, 1)$, we say a graph $G = (V, E)$ is an *α-sampler* if for every $S \subseteq V$,

$$\Pr_{v\leftarrow V}\left[\left|\frac{|S|}{|V|} - \frac{\Gamma(v) \cap S}{\Gamma(v)}\right| > \alpha\right] < \alpha,$$

where $\Gamma(v)$ is the set of neighbours of v in G. By [Gol11], there is a $\mathrm{poly}(n)$-time algorithm that constructs a $(4/\alpha^4)$-regular graph on n vertices that is an α-sampler, given n and $\alpha \in (0,1)$. The new PCPP verifier works as follows:

- Let $\mathsf{seed} \in \{0,1\}^r$ be the random bits and $i \in [q]$ be the index of a query. If $V^{\mathsf{old}}(\mathsf{seed}, i)$ makes a query to the input oracle, it firstly makes the same query to the input oracle, and then probes $\Pi^{\mathsf{new}}(\mathsf{seed}, i)$ for Δ times. The last Δ queries to $\Pi^{\mathsf{new}}(\mathsf{seed}, i)$ are for the smoothness property.

- Now we assume that $V^{\mathsf{old}}(\mathsf{seed}, i)$ makes a query to the proof oracle. Let $\mathsf{NList} := \mathsf{NList}(\mathsf{seed}, k)$ be the ordered list of neighbours of (seed, k) from the rectangular neighbour listing property, $m := |\mathsf{NList}|$, and $\mathsf{self} \in [m]$ be the index of (seed, i) in the list. Let $\alpha := \mu/(10q)$, and $G_\alpha^{\mathsf{NList}}$ be an explicit $(\Delta - 1)$-regular α-sampler with m nodes from [Gol11]. Let $j_2, j_3, \ldots, j_\Delta \in [m]$ be the neighbours of self in $G_\alpha^{\mathsf{NList}}$, and $j_1 := \mathsf{self}$. The verifier probes

  $$\Pi^{\mathsf{new}}(\mathsf{NList}[j_1]), \Pi^{\mathsf{new}}(\mathsf{NList}[j_2]), \ldots, \Pi^{\mathsf{new}}(\mathsf{NList}[j_\Delta]),$$

  and rejects if the answers are not the same. Otherwise, the verifier treats the consistent answer as the answer to the i-th query of V^{old} and simulates V^{old}.

Rectangularity. Recall that the proof oracle is $\Pi^{\mathsf{new}} : \{0,1\}^{2^r} \times [q] \to \{0,1\}$, where $r = r_{\mathsf{row}} + r_{\mathsf{col}} + r_{\mathsf{shared}}$ is the randomness complexity. By the rectangular neighbour listing property of V^{old}, we know that the shared randomness can be partitioned into $(\mathsf{seed.shared.row}, \mathsf{seed.shared.col}) \in \{0,1\}^{r_{\mathsf{shared}}/2} \times \{0,1\}^{r_{\mathsf{shared}}/2}$. We define $W^{\mathsf{new}} := 2^{r_{\mathsf{col}} + r_{\mathsf{shared}}/2}$, $H^{\mathsf{new}} := q \cdot 2^{r_{\mathsf{row}} + r_{\mathsf{shared}}/2}$, and the $H^{\mathsf{new}} \times W^{\mathsf{new}}$ proof matrix

$$\begin{aligned}
\Pi^{\mathsf{new}}[u, v] &:= \Pi^{\mathsf{new}}(\mathsf{seed}, i); \\
\text{where} \quad u &:= (\mathsf{seed.row}, \mathsf{seed.shared.row}, i) \in \{0,1\}^{r_{\mathsf{row}} + r_{\mathsf{shared}}/2 + \log q}, \\
v &:= (\mathsf{seed.col}, \mathsf{seed.shared.col}) \in \{0,1\}^{r_{\mathsf{col}} + r_{\mathsf{shared}}/2}, \\
\mathsf{seed} &:= (\mathsf{seed.row}, \mathsf{seed.col}, \mathsf{seed.shared} := (\mathsf{seed.shared.row}, \mathsf{seed.shared.col})).
\end{aligned}$$

Now it suffices to construct the type predicate $V_{\mathsf{type}}^{\mathsf{new}}$ and the row and column verifiers $V_{\mathsf{row}}^{\mathsf{new}}$ and $V_{\mathsf{col}}^{\mathsf{new}}$. Recall that the new verifier V^{new} simulates V^{old} as follows: If V^{old} makes a query to the proof oracle, it makes Δ queries to the proof oracle using the RNL property; otherwise, it makes the same query to the input oracle and Δ queries to the same bit of the proof oracle.

- The type predicate $V_{\mathsf{type}}^{\mathsf{new}}$ (given the shared randomness) calls the type predicate $V_{\mathsf{type}}^{\mathsf{old}}$ of the old PCPP verifier, obtains the list of types of the queries, replaces each "proof" by Δ continuous "proof" and replaces each "input" by an "input" and Δ continuous "proof".

- For a query of V^{old} to the proof oracle, the row verifier $V_{\mathsf{row}}^{\mathsf{new}}$ (resp. the column verifier $V_{\mathsf{col}}^{\mathsf{new}}$) calls the row verifier $V_{\mathsf{row}}^{\mathsf{old}}$ (resp. the column verifier $V_{\mathsf{col}}^{\mathsf{old}}$) of the old PCPP. By the rectangular neighbour listing property, it can list the "row-part" (resp. the "column-part") of the neighbour list NList and know the index self in the list. It then constructs the sampler G_α, finds the Δ selected neighbours of self (including itself), and outputs the "row-parts" (resp. the "column-parts") of them.

- For a query of V^{old} to the input oracle, the row verifier $V_{\mathsf{row}}^{\mathsf{new}}$ (resp. the column verifier $V_{\mathsf{col}}^{\mathsf{new}}$) calls the row verifier $V_{\mathsf{row}}^{\mathsf{old}}$ (resp. the column verifier $V_{\mathsf{col}}^{\mathsf{old}}$) of the old PCPP to obtain the query to the input oracle rectangularly. It is easy to see that the remaining Δ queries to the proof oracle can be done rectangularly.

Smoothness. We need to show that for uniformly random $\mathsf{seed} \in \{0,1\}^r$ and $i \in [q \cdot \Delta]$, each bit of the proof oracle Π^{new} is equally likely to be probed given randomness seed on the i-th query. Let $i_1 := \lfloor (i-1)/\Delta \rfloor + 1$ and $i_2 := (i-1) \bmod \Delta + 1$. Let $\hat{G} = (\hat{V}, \hat{E})$ be the "union" of all $G_\alpha^{\mathsf{NList}}$, that is:

- $\hat{V} := \{0,1\}^{2^r} \times [q]$.
- Let $(\mathsf{seed}, i_1), (\mathsf{seed}', i_1')$ be two configurations given which V^{old} will probe the proof oracle. Then $((\mathsf{seed}, i_1), (\mathsf{seed}', i_1')) \in \hat{E}$ if and only if the configurations (seed, i_1) and (seed', i_1') are neighbours, and there is an edge between them in $G_\alpha^{\mathsf{NList}}$, where NList is the neighbourhood containing these two configurations.
- For each (seed, i_1) such that $V^{\mathsf{old}}(\mathsf{seed}, i_1)$ probes the input oracle, we add Δ self-loops on the node $(\mathsf{seed}, i_1) \in \hat{V}$.

Assume that $\mathsf{seed} \in \{0,1\}^r$ and $i \in [q \cdot \Delta]$ are uniformly chosen. The query pattern of V^{new} to the proof oracle is as follows: It firstly selects a node $(\mathsf{seed}, i_1) \in \hat{V}$ uniformly, and then chooses a uniform neighbour of it. It is easy to see that each bit of the proof oracle is probed with probability

$$\frac{\Delta}{2^r \cdot q \cdot \Delta} = \frac{1}{2^r \cdot q}.$$

Soundness. The soundness of V^{old} follows from [BHPT20, Appendix A.1] (which is for PCP instead of PCPP); for completeness, we present a self-contained proof here. Assume that x is δ-far from being in L and $\Pi^{\mathsf{new}} : \{0,1\}^{2^r} \times [q]$ is a proof, we need to show that the verifier accepts with probability at most $s + \mu$. Let $\Pi^{\mathsf{old}} : \{0,1\}^\ell \to \{0,1\}$ be defined as follows:

$$\Pi^{\mathsf{old}}[j] := \underset{(\mathsf{seed}, i) \in \{0,1\}^r \times [q]}{\mathsf{Majority}} \left\{ \Pi^{\mathsf{new}}(\mathsf{seed}, i) : V^{\mathsf{old}}(\mathsf{seed}, i) = j \right\},$$

By the soundness of V^{old}, we know that V^{old} will accept (x, Π^{old}) with probability at most s.

Let $\mathsf{idx}^i(\mathsf{seed}) \in [\ell]$ be the i-th query of V^{old} given randomness seed. An index $j \in [\ell]$ is said to be β-*consistent* if for at least β fraction of (seed, i) such that $\mathsf{idx}^i(\mathsf{seed}) = j$, $\Pi^{\mathsf{new}}(\mathsf{seed}, i) = \Pi^{\mathsf{old}}[j]$. We define the following events over the random variable seed:

- H is the event that for every $i \in [q]$, $\mathsf{idx}^i(\mathsf{seed})$ is $(1-2\alpha)$-consistent (recall that $\alpha := \mu/(10q)$ is the parameter of the sampler).
- M is the event that for every $i \in [q]$, $\Pi^{\mathsf{new}}(\mathsf{seed}, i) = \Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed}))$.
- A is the event that V^{new} accepts (x, Π^{new}) on the randomness seed.
- C_i is the event that the Δ queries made by V^{new} corresponding to the i-th query of V^{old} returns the same answer (i.e. the "consistency check" passes on the simulation of the i-th query of V^{old}).

Claim 6.4. $\Pr[A \wedge \overline{H}] \le q\alpha$.

Claim 6.5. $\Pr[\overline{M} \wedge H] \le 2q\alpha$.

Claim 6.6. $\Pr[A \wedge H] \le 2q\alpha + s$.

From the claims above we can see that

$$\begin{aligned}
\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[A] &= \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[A \wedge \overline{H}] + \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[A \wedge H] \\
&\le s + q\alpha + 2q\alpha \\
&\le s + \mu.
\end{aligned}$$

Proof of Claim 6.4. Let (seed, i) be a configuration, $\mathsf{NList} := \mathsf{NList}(\mathsf{seed}, i)$ be the list of all its neighbours, $G_\alpha^{\mathsf{NList}} = (V, E)$ be the explicit sampler graph corresponding to the neighbourhood NList (i.e. V contains the configurations in NList), and self be the node corresponding to (seed, i). We say that a configuration (seed, i) is an *error configuration* if

$$\left| \frac{|S|}{|V|} - \frac{|\Gamma(\mathsf{self}) \cap S|}{|\Gamma(\mathsf{self})|} \right| > \alpha,$$

where $S := \{(\mathsf{seed}', i') \in V \mid \Pi^{\mathsf{new}}(\mathsf{seed}', i') \neq \Pi^{\mathsf{new}}(\mathsf{seed}, i)\}$. Since $G_\alpha^{\mathsf{NList}}$ is an α-sampler, there are at most α fraction of error configurations in each neighbourhood. Suppose $\overline{H}$ happens, then there is some i such that $\mathsf{idx}^i(\mathsf{seed})$ is not $(1-2\alpha)$-consistent, which means that $|S|/|V| \geq 2\alpha$. Suppose in addition that C_i happens (i.e., the "consistency check" on the i-th query passes), then $\Gamma(\mathsf{self}) \cap S = \varnothing$, which means that (seed, i) is an error configuration.

Let E be the set of error configurations. We can then calculate that

$$\begin{aligned}
\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}[A \wedge \overline{H}] \leq & \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}\left[C_1 \wedge \cdots \wedge C_q \wedge \overline{H}\right] \\
\leq & \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}\left[\exists i \in [q]\ (\mathsf{seed}, i) \in E \wedge \overline{H}\right] \\
\leq & q \cdot \Pr_{\substack{\mathsf{seed} \leftarrow \{0,1\}^r \\ i \leftarrow [q]}}[(\mathsf{seed}, i) \in E \wedge \overline{H}] \\
\leq & q \cdot \alpha.
\end{aligned}$$

◇

Proof of Claim 6.5. For every $j \in [q]$, we denote H_j to be the event that $\mathsf{idx}^j(\mathsf{seed})$ is $(1-2\alpha)$-consistent (and thus $H = \bigwedge_{j \in [q]} H_j$). Let $H^\star$ be the event that $\mathsf{idx}^i(\mathsf{seed})$ is $(1-2\alpha)$-consistent over the random variable $(\mathsf{seed}, i) \in \{0,1\}^r \times [q]$. We can see that:

$$\begin{aligned}
\Pr_{\mathsf{seed} \leftarrow \{0,1\}^r}\left[\overline{M} \wedge H\right] \leq & q \cdot \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r, i \in [q]}\left[\Pi^{\mathsf{new}}(\mathsf{seed}, i) \neq \Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed})) \ \wedge \ \bigwedge_{j \in [q]} H_j\right] \\
\leq & q \cdot \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r, i \in [q]}\left[\Pi^{\mathsf{new}}(\mathsf{seed}, i) \neq \Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed})) \ \wedge \ H_i\right] \\
= & q \cdot \Pr_{\mathsf{seed} \leftarrow \{0,1\}^r, i \in [q]}\left[\Pi^{\mathsf{new}}(\mathsf{seed}, i) \neq \Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed})) \ \wedge \ H^\star\right]. \qquad (7)
\end{aligned}$$

Let N be the set of all neighbourhoods that contain a configuration $(\mathsf{seed}, i) \in H^\star$ (i.e. $\mathsf{idx}^i(\mathsf{seed})$ is $(1-2\alpha)$-consistent). By the definition of $H^\star$ and the neighbours of configurations, we can see that for each $h \in N$, all the configurations in h are also in $H^\star$. Thus the uniform distribution over $H^\star$ is identical to the following distribution: we first sample a neighbourhood $h \in N$ (with probability proportional to the size of h), then uniformly sample a configuration $(\mathsf{seed}, i) \in h$. Thus we have:

$$\begin{aligned}
(7) \leq & q \cdot \Pr_{(\mathsf{seed}, i) \leftarrow \{0,1\}^r \times [q]}\left[\Pi^{\mathsf{new}}(\mathsf{seed}, i) \neq \Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed})) \ \middle| \ H^\star\right] \\
= & q \cdot \mathop{\mathbb{E}}_{h \leftarrow \mathcal{N}}\left[\Pr_{(\mathsf{seed}, i) \leftarrow h}\left[\Pi^{\mathsf{new}}(\mathsf{seed}, i) \neq \Pi^{\mathsf{old}}(\mathsf{idx}^i(\mathsf{seed}))\right]\right] \\
\leq & q \cdot \mathop{\mathbb{E}}_{h \in \mathcal{N}}[2\alpha], \qquad (8) \\
\leq & 2q\alpha,
\end{aligned}$$

where $\mathcal{N}$ is some distribution over N, and Eq. (8) holds by the definition of $(1-2\alpha)$-consistency. ◇

Proof of Claim 6.6. We can see that:

$$\begin{aligned}\Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[A\wedge H] &\le \Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[\overline{M}\wedge H]+\Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}[A\wedge H\wedge M]\\ &\le 2q\alpha+\Pr_{\mathsf{seed}\leftarrow\{0,1\}^r}\left[V^{\mathsf{old}}\text{ accepts }(x,\Pi^{\mathsf{old}})\wedge H\wedge M\right]\\ &\le 2q\alpha+\Pr_{\mathsf{seed}\in\{0,1\}^r}\left[V^{\mathsf{old}}\text{ accepts }(x,\Pi^{\mathsf{old}})\right]\\ &\le 2q\alpha+s.\end{aligned}$$

Note that the first inequality follows from the definition of V^{new} and Π^{old}. ◇

Other Properties. The query complexity, parity-check complexity, and decision complexity can be easily checked by the definition. □

6.2 A Rectangular PCPP with RNL Property

In this section, we construct a rectangular PCPP with rectangular neighbour listing property, by combining [BHPT20] and [RSW22, Section 7.1].

Lemma 6.7 (Lemma 7.3 of [RSW22])**.** *Let $\lambda < 0.1$, q, m be integers such that $q \ge \log\frac{4}{\lambda}$, 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 $O((qm/\lambda)^2)$.*

Theorem 6.8. *For all constants $\delta > 0$, there is a constant $\rho \in (0,1)$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \le m \le (\log T(n))^{0.1}$, $n \le T(n) \le 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \le \log T(n)$, and $w_{\mathsf{input}}(n) \le \log n$. Then there are good functions $h_{\mathsf{proof}}(n)$ and $h_{\mathsf{input}}(n)$ satisfying*

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

such that the following holds.

Suppose that $h_{\mathsf{proof}}, w_{\mathsf{proof}} \ge (4/m)\log T(n)$, and that for some absolute constant $C \ge 1$,[20]

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

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 neighbour listable, 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 4.

[20] Note that in this theorem, we allow m to be a super-constant. Plugging in the inequalities $w_{\mathsf{input}} \le k$ and $k + h_{\mathsf{input}} \le hm$ in [RSW22, Section 7.1.2, Eq. (7)], it turns out that both $\frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}}$ and $\frac{h_{\mathsf{input}}}{h_{\mathsf{proof}}}$ needs to be at most $1 - \frac{Cm\log\log T}{\log T}$ instead of simply $1 - \frac{C\log\log T}{\log T}$.

Soundness error	$1-\rho$
Proximity parameter	δ
Robustness parameter	ρ
Row randomness	$h_{\mathsf{proof}} - (4/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (4/m)\log T(n)$
Shared randomness	$(7/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	$T(n)^{1/m}\cdot \mathrm{polylog}(T(n))$
Decision complexity	
RNL time complexity	$\mathrm{poly}(\log T(n), m^m)$

Table 4: Parameters of the PCPP constructed in Theorem 6.8.

Basic Definitions. Let α, t, h, f be defined as in [RSW22, Section 7.1.1], where: α is a universal constant, $t := \log T(n)$, $h := \lceil (t+3)/m \rceil$, $f := h + \alpha \log_2 t$. Let $\lambda := \min\{1/(ct), 1/m^{2cm}\}$ for some universal constant c.[21] We work with the field $\mathbb{F} := \mathrm{GF}(2^f)$.

By field theory, we know that $\mathbb{F}$ is an f-dimensional vector space over $\mathbb{F}_2$. Let $e_1, e_2, \ldots, e_f$ be a basis, then each element in $\mathbb{F}$ can be uniquely represented by $\sum_{i=1}^{f} e_i b_i$ for $(b_1, b_2, \ldots, b_f) \in \{0,1\}^f$. Let $H := \mathsf{span}\{e_1, e_2, \ldots, e_h\}$. Let $\mathsf{bin}_{H^m} : H^m \to \{0,1\}^{hm}$ and $\mathsf{bin}_{\mathbb{F}^m} : \mathbb{F}^m \to \{0,1\}^{fm}$ be the two bijections in [RSW22, Section 7.1.1], where

$$\mathsf{bin}_{\mathbb{F}^m} : \left(\sum_{i=1}^{f} e_i b_i, \sum_{i=1}^{f} e_i b_{f+i}, \ldots, \sum_{i=1}^{f} e_i b_{(m-1)f+i}\right) \mapsto (b_1, b_2, \ldots, b_{mf}).$$

The bijection bin_{H^m} will be used to define the queries to the input oracle (see [RSW22, Section 7.1.2]), which is not needed for our purpose of verifying the rectangular neighbour listing property.

We identify the binary strings as numbers where the leftmost bit is the least significant bit. Let $I_t : [n] \to H^m$ be the map to project the input to H^m defined as $I_t(i) := \mathsf{bin}_{H_m}^{-1}(2^{t+1} + i)$, and $I := \{I_t(k) : k \le |\Pi_{\mathsf{input}}|\}$.

The Proof. The PCPP proof is a Boolean string of length $\ell \cdot |\mathbb{F}|^m$ for some $\ell = \mathrm{polylog}(T)$. We consider the PCPP proof as an oracle $\Pi_{\mathsf{proof}} : \mathbb{F}^m \to \{0,1\}^\ell$, where the i-th bit of the proof is the j'-th bit of $\Pi_{\mathsf{proof}}[\mathsf{bin}_{\mathbb{F}^m}^{-1}(j)]$ for $j := \lfloor i/\ell \rfloor$ and $j' := i \bmod \ell$.[22] Without loss of generality we assume that ℓ is a power of 2. Note that to ensure rectangularity, the actual proof is treated as an $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$ Boolean matrix, where $H_{\mathsf{proof}} = \ell \cdot |\mathbb{F}|^m / W_{\mathsf{proof}}$ and thus

$$\begin{aligned} h_{\mathsf{proof}} &= \log \ell + m \log |\mathbb{F}| - w_{\mathsf{proof}} \\ &= O(\log\log T(n)) + mh + m \cdot \alpha \log\log T(n) - w_{\mathsf{proof}} \\ &= \log T(n) + \Theta(m \log\log T(n)) - w_{\mathsf{proof}}. \end{aligned}$$

The layout of the proof matrix which will be discussed later.

Query Pattern. The *line* over $\mathbb{F}^m$ with *intercept* $\vec{x} \in \mathbb{F}^m$ and *direction* $\vec{y} \in \mathbb{F}^m$ is the set $\{\vec{x} + t\vec{y} : t \in \mathbb{F}\}$, denoted by $\vec{x} + \mathbb{F}\vec{y}$. We now describe the query pattern of the PCPP verifier.

[21] In [RSW22, Section 7.1.2] they set $\lambda = 1/(ct)$, since when m is a constant $1/(ct) \le 1/m^{2cm}$ always holds in sufficiently large n. We set λ as this to satisfy the technical requirement in the soundness proof, which will be discussed later.

[22] The ℓ-bit output of Π_{proof} is actually a word of a non-Boolean PCPP encoded by some linear-time computable error-correcting code, see [BHPT20, Section 5.2]. We omit the details as it is not important to us.

Let $S_\lambda \subseteq \mathbb{F}^m$ be the explicit λ-biased set in Lemma 6.7 and seed be the randomness of the verifier, defined as $\mathsf{seed} := (R_2, R_3, \ldots, R_m, R_y) \in \mathbb{F}^{m-1} \times [|S_\lambda|]$. Note that $|S_\lambda| = O((fm/\lambda)^2)$. We define $\vec{x} := (0, R_2, R_3, \ldots, R_m)$, $\vec{y}_e := (1, 0, \ldots, 0)$, $\vec{y} := S_\lambda[R_y]$, $\mathcal{L}_0 := \vec{x} + \mathbb{F}\vec{y}_e$, and $\mathcal{L}_1 := \vec{x} + \mathbb{F}\vec{y}$. Denote

$$\mathsf{shift}(x_1, x_2, \ldots, x_m) := (x_2, x_3, \ldots, x_m, x_1),$$

and $\mathsf{shift}(\mathcal{L}) := \{\mathsf{shift}(\vec{x}) : x \in \mathcal{L}\}$ for a line $\mathcal{L}$. The PCPP verifier will query the following locations (see [BHPT20, Section C]):

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

It is easy to see that the type predicate V_{type} is pretty simple: the first $3\ell \cdot |\mathbb{F}|$ queries are to the proof (i.e., the first $3|\mathbb{F}|$ queries to Π_{proof} for $\mathcal{L}_0$, $\mathsf{shift}(\mathcal{L}_0)$, and $\mathcal{L}_1$), and the remaining $|\mathbb{F}|$ queries are to the input (i.e., the $|\mathbb{F}|$ queries to $\mathcal{L}_1 \cap I$). To verify the rectangular neighbour listing property, we only need to consider the $3|\mathbb{F}|$ queries to the proof oracle. Denote these queries to be $(\vec{a}_1, \ldots, \vec{a}_{|\mathbb{F}|})$, $(\vec{a}_{|\mathbb{F}|+1}, \ldots, \vec{a}_{2|\mathbb{F}|})$, and $(\vec{a}_{2|\mathbb{F}|+1}, \ldots, \vec{a}_{3|\mathbb{F}|})$, for $\mathcal{L}_0$, $\mathsf{shift}(\mathcal{L})$, and $\mathcal{L}_1$, respectively.

Rectangularity of the Proof. Let $k := \lceil (w_{\mathsf{proof}} - \log \ell) \cdot (h/f) \rceil$ and $c_2 := \lceil k/h \rceil$. We partition the random bits $\mathsf{seed} = (R_2, R_3, \ldots, R_m, R_y) \in \mathbb{F}^{m-1} \times [|S_\lambda|]$ as follows:

$$\begin{aligned}
\mathsf{seed.col} &:= (R_3, R_4, \ldots, R_{c_2-1}), \\
\mathsf{seed.row} &:= (R_{c_2+2}, R_{c_2+3}, \ldots, R_{m-1}), \\
\mathsf{seed.shared} &:= (R_2, R_{c_2}, R_{c_2+1}, R_m, R_y).
\end{aligned}$$

Lemma 6.9 ([RSW22], Lemma 7.5)**.** *Let* $\mathsf{seed} = (R_2, R_3, \ldots, R_m, R_y)$ *as defined above,* $R_1 = R_{m+1} = 0^f$, $\vec{\alpha}_i = (a_{i,1}, \ldots, a_{i,m}) \in \mathbb{F}^m$ *be the queries. For every* $j \in [m]$, $(a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j})$ *can be efficiently computed given* R_j, R_{j+1}, *and* R_y.

Proof. Let $\mathbb{F} = \{h_1, h_2, \ldots, h_{|\mathbb{F}|}\}$ and $j \in [m]$. We can see that

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

where $\vec{y}_1 := (1, 0, \ldots, 0) \in \mathbb{F}^m$, $\vec{y}_2 := (0, \ldots, 0, 1) \in \mathbb{F}^m$, and $\vec{y}_3 = S_\lambda[R_y]$. □

By Lemma 6.9, we can see that: Given $\mathsf{seed.shared}$ and $\mathsf{seed.col}$, we can recover $a_{1,j}, \ldots, a_{3|\mathbb{F}|,j}$ for every $j \in [1, c_2]$; given $\mathsf{seed.shared}$ and $\mathsf{seed.row}$, we can recover $a_{1,j}, a_{2,j}, \ldots, a_{3|\mathbb{F}|,j}$ for every $j \in [c_2, m]$. Note that

$$\begin{aligned}
|\mathsf{seed.col}| &= (c_2 - 3)f \geq w_{\mathsf{proof}} - 4t/m \geq 0, \\
|\mathsf{seed.row}| &= (m - c_2 - 2)f \geq h_{\mathsf{proof}} - 4t/m \geq 0.
\end{aligned}$$

We assume without loss of generality that t/m is an integer. Similar to [RSW22, Section 7.1.2], we move some bits to the shared randomness so that $|\mathsf{seed.col}| = w_{\mathsf{proof}} - 4t/m$ and $|\mathsf{seed.row}| = h_{\mathsf{proof}} - 4t/m$, then

$$\begin{aligned}
|\mathsf{seed.shared}| &= (m-1)f + |R_y| - (w_{\mathsf{proof}} + h_{\mathsf{proof}}) + 8t/m. \\
&\leq (m-1)f + O(\log(fm/\lambda)) - (\log \ell + mf) + 8t/m \\
&\leq 8t/m + O(\log t + m \log m) - f \\
&\leq 7t/m + O(\log t + m \log m).
\end{aligned}$$

Let $i \in [3\ell \cdot |\mathbb{F}|]$ be an index, $j := \lfloor (i-1)/\ell \rfloor + 1$, and $j' := (i-1) \bmod \ell$. The i-th query asks for the j'-th bit of $\Pi_{\mathsf{proof}}[\vec{a}_j]$, or equivalently, the $(\mathsf{bin}_{\mathbb{F}^m}(\vec{a}_j) \cdot \ell + j')$-th bit of the proof. We will construct V_{row} and V_{col} such that

$$\begin{aligned} V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared}, i) &\to \mathsf{irow}[i], \\ V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared}, i) &\to \mathsf{icol}[i], \\ \mathsf{irow}[i] \cdot W_{\mathsf{proof}} + \mathsf{icol}[i] &= \mathsf{bin}_{\mathbb{F}^m}(\vec{a}_j) \cdot \ell + j'. \end{aligned}$$

That is, if we place the proof into a matrix such that the (p, q)-th entry represents the $p \cdot W_{\mathsf{proof}} + q$ bit of the proof, the first $3\ell \cdot |\mathbb{F}|$ queries to the proof can be made rectangular.

- $V_{\mathsf{col}}(\mathsf{seed.col}, \mathsf{seed.shared}, i)$ computes $a_{j,1}, a_{j,2}, \ldots, a_{j,c_2}$ by Lemma 6.9; it outputs $\mathsf{icol}[i]$ as the concatenation of j' and $a_{j,1}, a_{j,2}, \ldots, a_{j,c_2-1}$ and the lowest $w_{\mathsf{proof}} - (c_2 - 1)f - \log \ell$ bits of a_{j,c_2}.
- $V_{\mathsf{row}}(\mathsf{seed.row}, \mathsf{seed.shared}, i)$ computes $a_{j,c_2}, a_{j,c_2+1}, \ldots, a_{j,m}$ by Lemma 6.9; 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 $a_{j,c_2+1}, a_{j,c_2+1}, \ldots, a_{j,m}$.

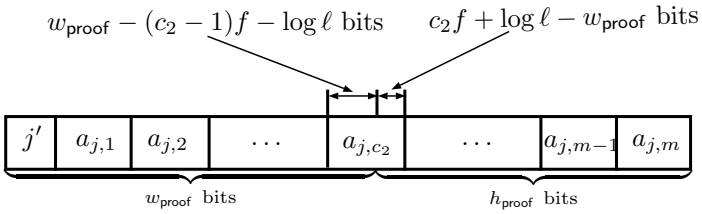

Figure 5: The binary representation of $\mathsf{bin}_{\mathbb{F}^m}(\vec{a}_j) \cdot \ell + j'$, where the leftmost bits are the least significant ones.

Recall that we identify binary strings and numbers with the leftmost bit as the least significant bit. It is then easy to see that $\mathsf{irow}[i] \cdot W_{\mathsf{proof}} + \mathsf{icol}[i]$ is the concatenation of $\mathsf{irow}[i]$ and $\mathsf{icol}[i]$, and $\mathsf{bin}_{\mathbb{F}^m}(\vec{a}_j) \cdot \ell + j'$ is the concatenation of j' and $\mathsf{bin}_{\mathbb{F}^m}(\vec{a}_j)$. Both of them are the concatenation of $j', a_{j,1}, a_{j,2}, \ldots, a_{j,m}$, hence $\mathsf{irow}[i] \cdot W_{\mathsf{proof}} + \mathsf{icol}[i] = \mathsf{bin}_{\mathbb{F}^m}(\vec{a}_j) \cdot \ell + j'$. Similarly, we arrange the input matrix as in [RSW22] so that the queries to the input oracle are also rectangular.

neighbours of (seed, i). Let $\mathbb{F} = \{h_1, h_2, \ldots, h_{|\mathbb{F}|}\}$. Let $\mathsf{seed} = (R_2, R_3, \ldots, R_y)$, $i \in [3\ell \cdot |\mathbb{F}|]$, $j := \lfloor (i-1)/\ell \rfloor + 1$, and $j' := (i-1) \bmod \ell$. Assume that this query probes the j'-th bit of $\Pi_{\mathsf{proof}}[\vec{a}_j]$, where $\vec{a}_j = (a_{j,1}, a_{j,2}, \ldots, a_{j,m})$. We define the *canonical neighbour* (seed_1, i_1) of (seed, i) as follows:

$$\mathsf{seed}_1 := (R_2^1 := a_{j,2}, R_3^1 := a_{j,3}, \ldots, R_m^1 := a_{j,m}, R_y^1 := 0) \tag{9}$$

$$i_1 := (j_1 - 1) \cdot \ell + j_1' + 1 \tag{10}$$

where $j_1 \in [1, |\mathbb{F}|]$ such that $h_{j_1} = a_{j,1}$, and $j_1' = j'$. It is easy to see that the canonical neighbours are the representative elements of the equivalence class induced by the neighbourhood relation. Denote $\mathcal{S}$ as the set of canonical neighbours, then $(\mathsf{seed}, i) \in \mathcal{S}$ if and only if $i \in [1, |\mathbb{F}|]$ and $R_y = 0$.

To list all the neighbours of (seed, i), it suffices to find its canonical neighbour (seed_1, i_1) and list all the neighbours of (seed_1, i_1). Let $\mathsf{seed}_2 = (R_2^2, R_3^2, \ldots, R_m^2, R_y^2)$, $i_2 \in [3\ell \cdot |\mathbb{F}|]$, $j_2 := \lfloor (i_2 - 1)/\ell \rfloor + 1$, $j_2' := (i_2 - 1) \bmod \ell$. Suppose that (seed_2, i_2) is a neighbour of (seed_1, i_1), then they represent the

queries to the same bit of the same entry of Π_{proof}. This means that $j_1' = j_2'$, and one of the following conditions holds:

$$j_2 \in [1, |\mathbb{F}|] \quad \text{and} \quad (h_{j_2}, R_2^2, R_3^2, \ldots, R_m^2) = (h_{j_1}, R_2^1, \ldots, R_{m-1}^1, R_m^1) \tag{11}$$

$$j_2 \in [|\mathbb{F}|+1, 2|\mathbb{F}|] \quad \text{and} \quad (R_2^2, R_3^2, \ldots, R_m^2, h_{j_2-|\mathbb{F}|}) = (h_{j_1}, R_2^1, \ldots, R_{m-1}^1, R_m^1); \tag{12}$$

$$j_2 \in [2|\mathbb{F}|+1, 3|\mathbb{F}|] \quad \text{and} \quad (h \cdot y_1, R_2^2 + h \cdot y_2, \ldots, R_m^2 + h \cdot y_m) = (h_{j_1}, R_2^1, \ldots, R_m^1)$$
$$\text{where } h := h_{j_2-2|\mathbb{F}|}, (y_1, y_2, \ldots, y_m) := S_\lambda[R_y^2]. \tag{13}$$

We will list the neighbours of (seed_1, i_1) in the following order: the $|S_\lambda|$ configurations satisfying (11) in the lexicographic order of R_y^2 (note that this includes (seed_1, i_1)), the $|S_\lambda|$ configurations satisfying (12) in the lexicographic order of R_y^2, and then the $|S_\lambda|$ configurations satisfying (13) in the lexicographic order of R_y^2.[23]

Rectangular neighbour Listing of $\mathcal{S}$. Now we need to verify that the aforementioned listing of the neighbours satisfies the rectangular neighbour listing property (see Definition 6.2). To start with, we consider the case when $(\mathsf{seed}, i) = (\mathsf{seed}_1, i_1)$, i.e., $(\mathsf{seed}, i) \in \mathcal{S}$. Recall that

$$\begin{aligned}
\mathsf{seed}_1.\mathsf{col} &:= (R_3^1, R_4^1, \ldots, R_{c_2-1}^1), \\
\mathsf{seed}_1.\mathsf{row} &:= (R_{c_2+2}^1, R_{c_2+3}^1, \ldots, R_{m-1}^1), \\
\mathsf{seed}_1.\mathsf{shared} &:= (R_2^1, R_{c_2}^1, R_{c_2+1}^1, R_m^1, R_y^1).
\end{aligned}$$

Let $\mathsf{low}(\cdot)$ and $\mathsf{high}(\cdot)$ denote the lower and higher halves of a Boolean string, respectively. We partition $\mathsf{seed.shared} = (R_2^1, R_{c_2}^1, R_{c_2+1}^1, R_m^1, R_y^1)$ into two parts $(\mathsf{seed.shared.row}, \mathsf{seed.shared.col})$, where $\mathsf{seed.shared.col} := (R_2^1, R_{c_2}, \mathsf{low}(R_y))$ and $\mathsf{seed.shared.row} := (R_{c_2+1}, R_m^1, \mathsf{high}(R_y))$.

- The column algorithm $A_{\mathsf{col}}(\mathsf{seed}_1.\mathsf{col}, \mathsf{seed}_1.\mathsf{shared}, i_1)$ outputs $\mathsf{NList}_{\mathsf{col}}(\mathsf{seed}_1, i_1)$ as follows:

  1. It firstly outputs the column-part of the $|S_\lambda|$ neighbours (seed_2, i_2) satisfying (11), including (seed_1, i_1) itself. We can see that $i_2 = i_1$, $\mathsf{seed}_2.\mathsf{col} = \mathsf{seed}_1.\mathsf{col}$, and $\mathsf{seed}_2.\mathsf{shared} = (R_2^1, R_{c_2}^1, R_{c_2+1}^1, R_m^1, R_y^2)$, where R_y^2 enumerates over all $|S_\lambda|$ possibilities in lexicographic order.
  2. It then aims to output the column-part of the $|S_\lambda|$ neighbours (seed_2, i_2) satisfying (12). We will enumerate all R_y^2 in lexicographic order and output:

$$\begin{aligned}
\mathsf{seed}_2.\mathsf{col} &= (R_3^2 = R_2^1, \ldots, R_{c_2-1}^2 = R_{c_2-2}^1), \\
\mathsf{seed}_2.\mathsf{shared.col} &= (R_2^2 = h_{j_1}, R_{c_2}^2 = R_{c_2-1}^1, \mathsf{low}(R_y^2)).
\end{aligned}$$

     Let $j_2 := \lfloor (i_2 - 1)/\ell \rfloor + 1$ and $j_2' := (i_2 - 1) \bmod \ell$, we can see that $j_2' = j_1'$ and $h_{j_2} = R_m^1$, hence i_2 can also be computed efficiently.
  3. Finally, it aims to output the column-part of the neighbours (seed_2, i_2) satisfying (13). We enumerate R_y^2 in lexicographic order. Let $(y_1, y_2, \ldots, y_m) = S_\lambda[R_y^2]$. Denote $h := y_1^{-1} \cdot h_{j_1}$, and let j_2 be the unique number in $[2|\mathbb{F}|+1, 3|\mathbb{F}|]$ such that $h = h_{j_2-2|\mathbb{F}|}$, we output:

$$\begin{aligned}
i_2 &= (j_2 - 1) \cdot \ell + j_1' + 1 \\
\mathsf{seed}_2.\mathsf{col} &= (R_3^2 = R_3^1 - h \cdot y_3, \ldots, R_{c_2-1}^2 = R_{c_2-1}^1 - h \cdot y_{c_2-1}) \\
\mathsf{seed}_2.\mathsf{shared.col} &= (R_2^2 = R_2^1 - h \cdot y_2, R_{c_2}^2 = R_{c_2}^1 - h \cdot y_{c_2}, \mathsf{low}(R_y^2)).
\end{aligned}$$

[23] Recall that for every $(y_1, y_2, \ldots, y_m) = S_\lambda[R_y^2]$, $y_1 \neq 0$. Thus for every R_y^2, there is exactly one $j_2 \in [2|\mathbb{F}|+1, 3|\mathbb{F}|]$ that satisfies (13), namely the j_2 such that $h_{j_2-2|\mathbb{F}|} = y_1^{-1} \cdot h_{j_1}$.

- Similarly, the row algorithm $A_{\mathsf{row}}(\mathsf{seed}_1.\mathsf{row}, \mathsf{seed}_1.\mathsf{shared}, i_1)$ can output $\mathsf{NList}_{\mathsf{row}}(\mathsf{seed}_1, i_1)$. We omit the details since it can be adapted from A_{col} directly.

It is clear that the "zipped" list of $\mathsf{NList}_{\mathsf{row}}$ and $\mathsf{NList}_{\mathsf{col}}$ is the list of neighbours of (seed_1, i_1). Since (seed_1, i_1) appears at the head of the list, A_{row} and A_{col} can simply output $\mathsf{self}_{\mathsf{row}} = \mathsf{self}_{\mathsf{col}} = 1$.

Rectangular neighbour Listing for $\overline{\mathcal{S}}$. For the general case when $(\mathsf{seed}, i) \notin \mathcal{S}$, the algorithms A_{row} and A_{col} need to find its canonical neighbour (seed_1, i_1) "rectangularly": Let (seed_1, i_1) be the canonical neighbour of (seed, i), then A_{row} (resp. A_{col}) can output the row-part (resp. the column-part) of (seed_1, i_1) given $(\mathsf{seed}.\mathsf{row}, \mathsf{seed}.\mathsf{shared}, i)$ (resp. $(\mathsf{seed}.\mathsf{col}, \mathsf{seed}.\mathsf{shared}, i)$). This can be done by checking Equation (9), Equation (10), and Lemma 6.9.

Finally, we need to verify that both A_{row} and A_{col} know the index of (seed, i) itself in this list. Let $j := \lfloor (i-1)/\ell \rfloor + 1$ and $j' := (i-1) \bmod \ell$. The list of (seed_1, i_1) contains three parts: the neighbours specified by (11), (12), and (13). By checking j, which is known by both A_{row} and A_{col}, we can find the part that contains (seed, i). The index of (seed, i) within the part is then determined by the lexicographic order of R_y in $\mathsf{seed}.\mathsf{shared}$.

Complexity of A_{row} and A_{col}. Recall that $\log|\mathbb{F}| = f = (\log T(n) + 3)/m + O(\log\log T(n))$, $|S_\lambda| = O((mf/\lambda)^2) = \mathrm{poly}(m^m, \log T(n))$. It is then easy to check that the running time of both A_{row} and A_{col} is $\mathrm{poly}(m^{O(m)}, f, \log T(n)) = \mathrm{poly}(m^m, \log T(n))$.

Robust Soundness. Finally, we prove a *weak* version of robust soundness here. This PCPP only guarantees an *expected* version of 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. (See [BGH+06, Lemma 8.11].) We use a Markov bound to turn this into a standard robust soundness property, but only with soundness parameter very close to 1. Since the robust soundness amplification (Section 6.4) preserves smoothness but does not seem to preserve RNL, we do not apply it here.

Let $\delta_{\mathsf{proof}}(\mathsf{seed})$ (resp. $\delta_{\mathsf{input}}(\mathsf{seed})$) be the fraction of bits of Π_{proof} (resp. Π_{input}) read by the verifier that we need to flip to make the verifier accept given the randomness seed. Let $\hat{\delta}(\mathsf{seed})$ be the fraction of bits (of both Π_{proof} and Π_{input}) read by the verifier that we need to flip to make the verifier accept given the randomness seed. By [BGH+06, Lemma 8.11] (also see [RSW22, Proof of Theorem 7.1]),[24] there is a constant $\rho_0 \in (0,1)$ such that for every constant $\delta \in (0,1)$, if Π_{input} is δ-far from being in L, then for any proof oracle Π_{proof}, either $\mathbb{E}_{\mathsf{seed}}[\delta_{\mathsf{proof}}(\mathsf{seed})] \geq \rho_0$ or $\mathbb{E}_{\mathsf{seed}}[\delta_{\mathsf{input}}(\mathsf{seed})] \geq \delta/3$. Fix this constant ρ_0.

Recall that the verifier V^{new} makes $|\mathbb{F}|$ queries to Π_{input} and $3\ell \cdot |\mathbb{F}|$ queries to Π_{proof}. We repeat each query to the input oracle for $9(\rho_0/\delta)\ell$ times. Then if Π_{input} is δ-far from being in L, the fraction of bits read by the verifier that we need to flip on expectation to make the verifier accept is

$$\mathop{\mathbb{E}}_{\mathsf{seed}}\left[\hat{\delta}(\mathsf{seed})\right] = \frac{\min\{\rho_0 \cdot 3\ell \cdot |\mathbb{F}|, (\delta/3) \cdot 9(\rho_0/\delta)\ell \cdot |\mathbb{F}|\}}{3\ell \cdot |\mathbb{F}| + 9(\rho_0/\delta)\ell \cdot |\mathbb{F}|} \geq \frac{3\ell\rho_0}{3\ell + 9(\rho_0/\delta)\ell} \geq \frac{\rho_0}{1 + 3\rho_0/\delta}.$$

Let $\rho := \frac{\rho_0}{1+3\rho_0/\delta}$. By a Markov bound,

$$\Pr_{\mathsf{seed}}\left[\hat{\delta}(\mathsf{seed}) \leq \rho\right] \leq 1 - \rho.$$

[24] Recall that we assume $m \leq (\log T(n))^{0.1}$ and set $\lambda = \min\{1/(ct), 1/m^{2cm}\}$, so that $m^m \leq T(n)^{1/m^2}$ and $\lambda \leq \min\{1/(ct), 1/m^{cm}\}$, which satisfies the technical requirement of [BGH+06, Lemma 8.11].

Verifier	V^{out}	V^{in}	V^{comp}
Soundness error	$1-\varepsilon^{\mathsf{out}}$	$1-\varepsilon^{\mathsf{in}}$	$1-\varepsilon^{\mathsf{out}}\cdot\varepsilon^{\mathsf{in}}$
Proximity parameter	δ^{out}	δ^{in}	δ^{out}
Robustness parameter	ρ^{out}	-	-
Row randomness	$r^{\mathsf{out}}_{\mathsf{row}}$	-	$r^{\mathsf{out}}_{\mathsf{row}}$
Column randomness	$r^{\mathsf{out}}_{\mathsf{col}}$	-	$r^{\mathsf{out}}_{\mathsf{col}}$
Shared randomness	$r^{\mathsf{out}}_{\mathsf{shared}}$	r^{in}	$r^{\mathsf{out}}_{\mathsf{shared}}+r^{\mathsf{in}}$
Proof matrix height	$H^{\mathsf{out}}_{\mathsf{proof}}$	-	$H^{\mathsf{out}}_{\mathsf{proof}}+2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}}$
Proof matrix width	$W^{\mathsf{out}}_{\mathsf{proof}}$	-	$W^{\mathsf{out}}_{\mathsf{proof}}$
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}	$H^{\mathsf{out}}_{\mathsf{proof}}\cdot W^{\mathsf{out}}_{\mathsf{proof}}+2^{r^{\mathsf{out}}+r^{\mathsf{in}}}$

Table 5: 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 $r^{\mathsf{in}}(d^{\mathsf{out}}(n))$.

Thus, the PCPP verifier has robust soundness error $1-\rho$ with robustness parameter ρ and proximity parameter δ.

6.3 RNL-Preserving Composition Theorem

Now we verify that the composition theorem in [RSW22, Section 7.2] preserves the rectangular neighbour listing property, using essentially the same approach of [BHPT20, Section 7.2]. This will be used to reduce the number of queries of our rectangular PCPP with RNL property.

Theorem 6.10. *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 (not necessarily rectangular) PCPP verifier V^{in} with parameters specified in Table 5. Moreover, assume that $q^{\mathsf{in}}=O(1)$, $\rho^{\mathsf{out}}\ge\delta^{\mathsf{in}}$, $\ell^{\mathsf{in}}=2^{r^{\mathsf{in}}}$,[25] $W^{\mathsf{out}}_{\mathsf{proof}}$ is a power of 2, and $r^{\mathsf{out}}_{\mathsf{col}}\le\log W^{\mathsf{out}}_{\mathsf{proof}}\le r^{\mathsf{out}}_{\mathsf{col}}+r^{\mathsf{out}}_{\mathsf{shared}}$. Then $\mathsf{NTIME}[T(n)]$ has a rectangular PCPP verifier V^{comp} with parameters specified in Table 5.*

Furthermore, if V^{out} has $t^{\mathsf{out}}_{\mathsf{RNL}}(n)$-time rectangular neighbour listing property, then V^{comp} has $t_{\mathsf{RNL}}(n)$-time rectangular neighbour listing property, where $t_{\mathsf{RNL}}(n):=\mathrm{poly}(t^{\mathsf{out}}_{\mathsf{RNL}}(n),\ell^{\mathsf{in}},q^{\mathsf{in}},d^{\mathsf{in}})$.

The composed PCPP verifier. Assume that we have a robust and rectangular PCPP verifier V^{out} for $L\in\mathsf{NTIME}[T(n)]$ and a PCPP verifier V^{in} for Circuit-Eval$^{\perp}$. We now describe the composed PCPP verifier V^{comp} for L (also see [BGH+06, Section 2.4], [BHPT20, Section 7.2], [RSW22, Section 7.2]). In a nutshell, we will reduce the verification of the outer PCPP V^{out} to Circuit-Eval$^{\perp}$, where the circuit represents the decision predicate of V^{out} and the input consists of the input of L and the proof for the outer PCPP. As in [BHPT20, RSW22], we need to carefully arrange the proof matrix to maintain the rectangularity.

Assume that $\Pi^{\mathsf{out}}_{\mathsf{input}}$, $\Pi^{\mathsf{out}}_{\mathsf{proof}}$, and $\mathsf{seed}^{\mathsf{out}}$ are the proof matrix, the input matrix, and the random seed of V^{out}, respectively. The input matrix of V^{comp} is simply the input matrix of V^{out}, denoted

[25] 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 $2^{r^{\mathsf{in}}}$.

by Π^{in}. The proof of V^{comp} is the concatenation of $\Pi^{\mathsf{out}}_{\mathsf{proof}}$ and $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ for every $\mathsf{seed}^{\mathsf{out}} \in \{0,1\}^{r^{\mathsf{out}}}$, where each $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ is a PCPP proof for "V^{in} accepts $\mathsf{seed}^{\mathsf{out}}$." The random seed is $\mathsf{seed} := \mathsf{seed}^{\mathsf{out}} \circ \mathsf{seed}^{\mathsf{in}}$. The verifier V^{comp} works as follows:

1. Obtain the decision circuit $\mathsf{Dec}^{\mathsf{out}}$ and the list of query indices $I^{\mathsf{out}} \leftarrow V^{\mathsf{out}}(\mathsf{seed}^{\mathsf{out}})$ of V^{out}.
2. Use the inner PCPP to verify the following Circuit-Eval$^{\perp}$ instance $(C, \Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}}))$: the (explicit) circuit $C : \{0,1,\perp\}^{q^{\mathsf{out}}(n)} \times \{0,1\}^{\hat{r}^{\mathsf{out}}} \to \{0,1,\perp\}$ and the (implicit) input $\Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}}) \in \{0,1,\perp\}^{q^{\mathsf{out}}(n)} \times \{0,1\}^{\hat{r}^{\mathsf{out}}}$ are defined as follows:

$$C(u,v) := \mathsf{Dec}^{\mathsf{out}}(u, \mathsf{Dec}(v)),$$
$$\Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}}) := ((\Pi_{\mathsf{input}} \circ \Pi^{\mathsf{out}}_{\mathsf{proof}})|_{I^{\mathsf{out}}}, \mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})),$$

   where $(\mathsf{Enc}, \mathsf{Dec})$ is a linear-time encodable and decodable error-correcting code such that $\mathsf{Enc} : \{0,1\}^{r^{\mathsf{out}}} \to \{0,1\}^{\hat{r}^{\mathsf{out}}}$ is linear over GF(2).[26] Specifically:

   (a) The decision circuit $\mathsf{Dec}^{\mathsf{comp}}$ of V^{comp} is defined as the decision circuit $\mathsf{Dec}^{\mathsf{in}}$ of V^{in}.

   (b) The queries are sampled using $V^{\mathsf{in}}(\mathsf{seed}^{\mathsf{in}})$ for the Circuit-Eval$^{\perp}$ instance defined above with the proof $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$, i.e., we sample the queries $I^{\mathsf{in}} \leftarrow V^{\mathsf{in}}(\mathsf{seed}^{\mathsf{in}})$ and "redirect" them to the input oracle and the proof of the composed PCPP to obtain I^{comp}.[27]

Rectangularity of V^{comp}. We now verify the rectangularity of the composed PCPP verifier. Recall that: the proof $\Pi^{\mathsf{out}}_{\mathsf{proof}} \in \{0,1\}^{\ell^{\mathsf{out}}}$ of V^{out} is arranged as an $H^{\mathsf{out}}_{\mathsf{proof}} \times W^{\mathsf{out}}_{\mathsf{proof}}$ matrix, where the i-th row and the j-th column $\Pi^{\mathsf{out}}_{\mathsf{proof}}[i,j] := \Pi^{\mathsf{out}}_{\mathsf{proof}}[(i-1)\cdot W^{\mathsf{out}}_{\mathsf{proof}} + j]$; the inner proofs $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}}) \in \{0,1\}^{\ell^{\mathsf{in}}}$ for every $\mathsf{seed}^{\mathsf{out}} \in \{0,1\}^{r^{\mathsf{out}}}$.

Let $W^{\mathsf{in}}_{\mathsf{proof}} := \min\{W^{\mathsf{out}}_{\mathsf{proof}}/2^{r^{\mathsf{out}}_{\mathsf{col}}}, \ell^{\mathsf{in}}\}$ and $H^{\mathsf{in}}_{\mathsf{proof}} := \ell^{\mathsf{in}}/W^{\mathsf{in}}_{\mathsf{proof}}$. Note that both $W^{\mathsf{in}}_{\mathsf{proof}}$ and $H^{\mathsf{in}}_{\mathsf{proof}}$ are powers of 2, and $W^{\mathsf{out}}_{\mathsf{proof}} \geq 2^{r^{\mathsf{out}}_{\mathsf{col}}}$. We arrange the proof matrix as follows: The first $H^{\mathsf{out}}_{\mathsf{proof}}$ rows contain the $H^{\mathsf{out}}_{\mathsf{proof}} \times W^{\mathsf{out}}_{\mathsf{proof}}$ proof matrix of the outer PCPP; the rest part of the matrix is divided into blocks of size $W^{\mathsf{in}}_{\mathsf{proof}} \times H^{\mathsf{in}}_{\mathsf{proof}}$, each of which contains a proof of the inner PCPP $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ for some $\mathsf{seed}^{\mathsf{out}} \in \{0,1\}^{\ell^{\mathsf{in}}}$, sorted in the lexicographic order of $\mathsf{seed}^{\mathsf{out}}$. Clearly, the proof matrix height of the composed PCPP verifier is

$$H^{\mathsf{out}}_{\mathsf{proof}} + \ell^{\mathsf{in}} \cdot 2^{r^{\mathsf{out}}}/W^{\mathsf{out}}_{\mathsf{proof}} = H^{\mathsf{out}}_{\mathsf{proof}} + 2^{r^{\mathsf{out}}+r^{\mathsf{in}}}/W^{\mathsf{out}}_{\mathsf{proof}}.$$

Recall that the seed of the composed PCPP verifier is $\mathsf{seed} := (\mathsf{seed}^{\mathsf{out}}, \mathsf{seed}^{\mathsf{in}})$. Assume that the partition of random bits of V^{out} is $\mathsf{seed}^{\mathsf{out}} = (\mathsf{seed}^{\mathsf{out}}.\mathsf{row}, \mathsf{seed}^{\mathsf{out}}.\mathsf{col}, \mathsf{seed}^{\mathsf{out}}.\mathsf{shared})$. We partition the random bits as follows:

$$\mathsf{seed.shared} := (\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}, \mathsf{seed}^{\mathsf{in}}).$$
$$\mathsf{seed.row} := \mathsf{seed}^{\mathsf{out}}.\mathsf{row}.$$
$$\mathsf{seed.col} := \mathsf{seed}^{\mathsf{out}}.\mathsf{col}.$$

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

[27] In fact, there are three kinds of queries: the queries to $(\Pi_{\mathsf{input}} \circ \Pi^{\mathsf{out}}_{\mathsf{proof}})|_{I^{\mathsf{out}}}$, $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})$, and $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$. The queries of the first and the third kinds will be redirected as queries, and the second kind will be treated as a parity-check bit, since Enc is a linear function over GF(2). Details are contained in the verification of the rectangularity, also see [RSW22, Algorithm 2].

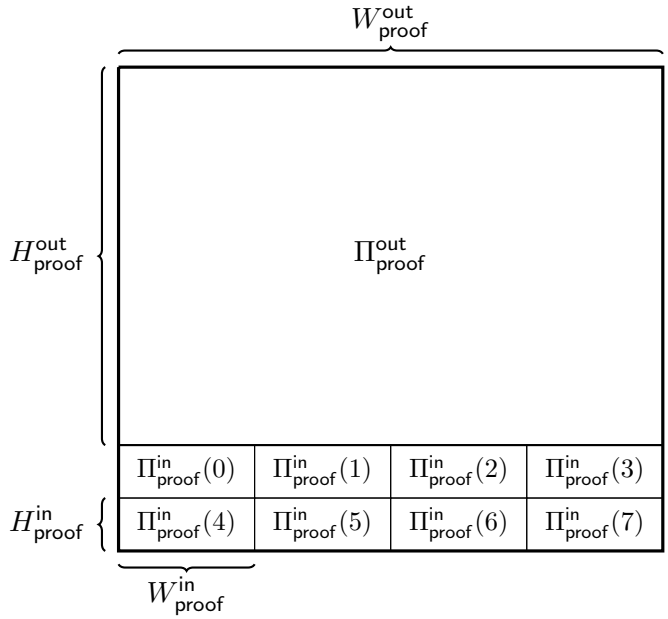

Figure 6: The layout of the proof matrix of the composed PCPP.

Now we describe the type predicate V_{type} and rectangular verifiers $V^{\mathsf{comp}}_{\mathsf{row}}$ and $V^{\mathsf{comp}}_{\mathsf{col}}$ of the composed PCPP.

The type predicate and the row/column verifier firstly obtain $\mathsf{seed}^{\mathsf{in}}$ in $\mathsf{seed.shared}$ and compute the queries $I^{\mathsf{in}} \leftarrow V^{\mathsf{in}}(\mathsf{seed}^{\mathsf{in}})$ of the inner PCPP for Circuit-Eval$^{\perp}$. There can be three cases for each query in I^{in}:

1. It probes the i-th cell of $\Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}})$ and $i \le q^{\mathsf{out}}(n)$, i.e., it queries $(\Pi_{\mathsf{input}} \circ \Pi^{\mathsf{out}}_{\mathsf{proof}})|_{I^{\mathsf{out}}}$.
   - The type predicate can call the type predicate of V^{out} to compute the type of the query, since it has $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$ and the index of the query in hand.
   - The row/column verifier of the composed PCPP runs the row/column verifier of the outer PCPP to obtain the row/column index of the query in $(\Pi_{\mathsf{input}}, \Pi^{\mathsf{out}}_{\mathsf{proof}})$. By the definition of $\Pi^{\mathsf{comp}}_{\mathsf{proof}}$, the row/column index is also the row/column index of the query in $(\Pi_{\mathsf{input}}, \Pi^{\mathsf{comp}}_{\mathsf{proof}})$.
2. It probes the i-th cell of $\Pi^{\mathsf{in}}_{\mathsf{input}}(\mathsf{seed}^{\mathsf{out}})$ and $i > q^{\mathsf{out}}(n)$, i.e., it queries $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})$. Instead of making a query, we fix this input of $\mathsf{Dec}^{\mathsf{comp}}$ to be $\mathsf{Enc}(\mathsf{seed}^{\mathsf{out}})[i - q^{\mathsf{out}}(n)]$. This bit will be considered as a parity-check bit.
3. It probes the i-th cell of $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$. This is a query to the proof, so the type predicate always outputs proof. Recall that $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ is placed in some $W^{\mathsf{in}}_{\mathsf{proof}} \times H^{\mathsf{in}}_{\mathsf{proof}}$ size block in the proof matrix. Let $N_{\mathsf{proof}} := W^{\mathsf{out}}_{\mathsf{proof}}/W^{\mathsf{in}}_{\mathsf{proof}}$, i.e., there are N_{proof} blocks of inner PCPP proofs in a row of the proof matrix. It is then easy to see that the column (resp. row) index of the query depends on $\mathsf{seed}^{\mathsf{in}}$ and the lowest $\log N_{\mathsf{proof}}$ bits (resp. the highest $r^{\mathsf{out}} - \log N_{\mathsf{proof}}$ bits) of $\mathsf{seed}^{\mathsf{out}}$. Note that since

$$\log N_{\mathsf{proof}} \ge \log\Big(W^{\mathsf{out}}_{\mathsf{proof}}/(W^{\mathsf{out}}_{\mathsf{proof}}/2^{r^{\mathsf{out}}_{\mathsf{col}}})\Big) \ge |\mathsf{seed}^{\mathsf{out}}.\mathsf{col}|,$$
$$\log N_{\mathsf{proof}} \le \log W^{\mathsf{out}}_{\mathsf{proof}} \le |\mathsf{seed}^{\mathsf{out}}.\mathsf{col}| + |\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}|,$$

we can arrange

$$\mathsf{seed}^{\mathsf{out}} := \mathsf{seed}^{\mathsf{out}}.\mathsf{col} \circ \mathsf{seed}^{\mathsf{out}}.\mathsf{shared} \circ \mathsf{seed}^{\mathsf{out}}.\mathsf{row}$$

such that the lowest $\log N_{\mathsf{proof}}$ bits (resp. the highest $r^{\mathsf{out}} - \log N_{\mathsf{proof}}$ bits) can be obtained from $\mathsf{seed}^{\mathsf{out}}.\mathsf{col}$ (resp. $\mathsf{seed}^{\mathsf{out}}.\mathsf{row}$) and $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$. Concretely, the row/column verifier will firstly identify the row/column index of the block that contains $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ and then obtain the row/column index of the query within the block by running the inner PCPP verifier $V^{\mathsf{in}}(\mathsf{seed}^{\mathsf{in}})$.

Rectangular neighbour Listing of V^{comp}. Now we verify that the composed PCPP verifier V^{comp} has the rectangular neighbour listing property with $t_{\mathsf{RNL}}(n) := \mathrm{poly}(t^{\mathsf{out}}_{\mathsf{RNL}}(n), \ell^{\mathsf{in}}, q^{\mathsf{in}}, d^{\mathsf{in}})$. Let (seed, k) be a configuration of V^{comp}, where

$$\mathsf{seed} = (\mathsf{seed}.\mathsf{row} := \mathsf{seed}^{\mathsf{out}}.\mathsf{row}, \mathsf{seed}.\mathsf{col} := \mathsf{seed}^{\mathsf{out}}.\mathsf{col}, \mathsf{seed}.\mathsf{shared} := (\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}, \mathsf{seed}^{\mathsf{in}})).$$

and $k \in [q^{\mathsf{comp}}]$. Assume that the verifier probes the proof matrix $\Pi^{\mathsf{comp}}_{\mathsf{proof}}$ on the k-th query given the randomness seed. By the discussion above, we know that the k-th query of the composed PCPP verifier can be one of the following two cases: a query to $\Pi^{\mathsf{out}}_{\mathsf{proof}}|_{I_{\mathsf{out}}}$ for $I_{\mathsf{out}} \leftarrow V^{\mathsf{out}}(\mathsf{seed}^{\mathsf{out}})$, or a query to $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$.

Assume that the rectangular neighbour listing algorithm for V^{out} partitions $\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}$ into $(\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}.\mathsf{row}, \mathsf{seed}^{\mathsf{out}}.\mathsf{shared}.\mathsf{col})$. We now partition $\mathsf{seed}.\mathsf{shared}$ as follows:

$$\mathsf{seed}.\mathsf{shared}.\mathsf{row} := (\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}.\mathsf{row}, \mathsf{low}(\mathsf{seed}^{\mathsf{in}})),$$
$$\mathsf{seed}.\mathsf{shared}.\mathsf{col} := (\mathsf{seed}^{\mathsf{out}}.\mathsf{shared}.\mathsf{col}, \mathsf{high}(\mathsf{seed}^{\mathsf{in}})).$$

The row and column algorithms A_{row} and A_{col} for the rectangular neighbour listing of V^{comp} work as follows.

Case 1. Given the configuration (seed, k), the verifier V^{comp} probes the i-th bit of $\Pi^{\mathsf{out}}_{\mathsf{proof}}|_{I_{\mathsf{out}}}$, where the index i depends on the $\mathsf{seed}^{\mathsf{in}}$. In other words, the composed PCPP verifier probes the answer of the i-th query made by the outer PCPP verifier when it is "simulating" the outer verifier using the inner PCPP verifier. A neighbour $(\mathsf{seed}' = (\mathsf{seed}'^{\mathsf{out}}, \mathsf{seed}'^{\mathsf{in}}), k')$ of (seed, k) must be a query of the same type, i.e., it is a query to the i'-th bit of $\Pi^{\mathsf{out}}_{\mathsf{proof}}|_{I'_{\mathsf{out}}}$ where the index i' depends on $\mathsf{seed}'^{\mathsf{in}}$. Furthermore, the i-th query index in I_{out} must be the same as the i'-th query index in I'_{out}. In such case, the row/column algorithms for rectangular neighbour listing will generate the following list:

1. The row algorithm $A^{\mathsf{comp}}_{\mathsf{row}}$ (resp. the column algorithm $A^{\mathsf{comp}}_{\mathsf{col}}$) firstly runs the outer PCPP verifier and generates the row part (resp. the column part) of I_{out}, and runs the inner PCPP verifier using $\mathsf{seed}^{\mathsf{in}}$ to obtain the index i defined above.
2. Then it runs the row algorithm $A^{\mathsf{out}}_{\mathsf{row}}$ (resp. the column algorithm $A^{\mathsf{out}}_{\mathsf{col}}$) for the outer PCPP to generate the row part (resp. the column part) of the list of neighbours of $(\mathsf{seed}^{\mathsf{out}}, i)$, denoted by L_{row} (resp. L_{col}).
3. For every $(\mathsf{seed}^{\mathsf{out}}_j.\mathsf{row}, \mathsf{seed}^{\mathsf{out}}_j.\mathsf{shared}.\mathsf{row}, i_j)$ (resp. $(\mathsf{seed}^{\mathsf{out}}_j.\mathsf{col}, \mathsf{seed}_j.\mathsf{shared}^{\mathsf{out}}.\mathsf{col}, i_j)$) in the list L_{row} (resp. L_{col}), we enumerate $(\mathsf{seed}'^{\mathsf{in}}, k') \in \{0,1\}^{r^{\mathsf{in}}} \times [q^{\mathsf{in}}]$ in lexicographic order. If the k'-th query of V^{in} given the $\mathsf{seed}'^{\mathsf{in}}$ as the seed is a query to the i'-th bit of $(\Pi_{\mathsf{input}} \circ \Pi^{\mathsf{out}}_{\mathsf{proof}})|_{I'_{\mathsf{out}}}$ (where i' depends on $\mathsf{seed}'^{\mathsf{in}}$), and $i' = i_j$, then append

$$(\mathsf{seed}^{\mathsf{out}}_j.\mathsf{row}, \mathsf{seed}^{\mathsf{out}}_j.\mathsf{shared}.\mathsf{row}, \mathsf{low}(\mathsf{seed}'^{\mathsf{in}}), k') \qquad (\text{for } A^{\mathsf{comp}}_{\mathsf{row}})$$
$$(\mathsf{seed}^{\mathsf{out}}_j.\mathsf{col}, \mathsf{seed}^{\mathsf{out}}_j.\mathsf{shared}.\mathsf{col}, \mathsf{high}(\mathsf{seed}'^{\mathsf{in}}), k') \qquad (\text{for } A^{\mathsf{comp}}_{\mathsf{col}})$$

to L_{row} (resp. L_{col}).

It is easy to check that the requirements of RNL property are satisfied.

Case 2. Given the configuration (seed, k), the verifier V^{comp} probes the i-th cell of $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$. Recall that for every $\mathsf{seed}^{\mathsf{out}} \in \{0,1\}^{r^{\mathsf{out}}}$, $\Pi^{\mathsf{in}}_{\mathsf{proof}}(\mathsf{seed}^{\mathsf{out}})$ is arranged in a block of size $H^{\mathsf{in}}_{\mathsf{proof}} \times W^{\mathsf{in}}_{\mathsf{proof}}$ in the proof matrix. The neighbours of (seed, k) need to query the same block, therefore the neighbours must have the same random seed for the outer PCPP verifier. Hence the row/column algorithms will work as follows:

1. The row algorithm $A^{\mathsf{comp}}_{\mathsf{row}}$ (resp. the column algorithm $A^{\mathsf{comp}}_{\mathsf{col}}$) firstly finds the list $\mathcal{L}^{\mathsf{in}} := \{(\mathsf{seed}^{\mathsf{in}}_j, k_j) \in \{0,1\}^{r^{\mathsf{in}}(d^{\mathsf{out}}(n))} \times [q^{\mathsf{comp}}]\}$ sorted in lexicographic order such that the inner PCPP will query the i_j-th bit of the inner proof on the k_j-th query given $\mathsf{seed}^{\mathsf{in}}_j$ as randomness. This can be done in $\mathrm{poly}(\ell^{\mathsf{in}}, q^{\mathsf{in}}, d^{\mathsf{in}})$-time by enumerating all possible $(\mathsf{seed}^{\mathsf{in}}_j, k_j)$ and running the inner PCPP verifier.
2. We define the final list of neighbours as
$$\mathcal{L} := \left\{ \left(\mathsf{seed}_j := (\mathsf{seed}^{\mathsf{out}}, \mathsf{seed}^{\mathsf{in}}_j, k_j) \right) : (\mathsf{seed}^{\mathsf{in}}_j, k_j) \in \mathcal{L}^{\mathsf{in}} \right\}.$$
It is easy to check that the list satisfies the promises of the rectangular neighbour listing property.

Other properties. The soundness error and proximity parameter can be found in [BGH+06, Section 2.4]. The query complexity, ROP parity-check complexity, and decision complexity can be found in [RSW22, Section 7.2]. We can see that the proof matrix of the composed PCPP verifier has width $W^{\mathsf{comp}}_{\mathsf{proof}} = W^{\mathsf{out}}_{\mathsf{proof}}$ and height $H^{\mathsf{comp}}_{\mathsf{proof}} = W^{\mathsf{out}}_{\mathsf{proof}} + 2^{r^{\mathsf{out}}(n) + r^{\mathsf{in}}(d^{\mathsf{out}}(n))} / W^{\mathsf{out}}_{\mathsf{proof}}$ (recall that $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$). By the definitions of the random seeds, we can see that: The row and column randomness complexity of V^{comp} is the same as the row and column randomness complexity of V^{out}, respectively; the shared randomness complexity of V^{comp} is the sum of the shared randomness complexity of V^{out} and the randomness complexity of V^{in}.

Remark 6.11. The composed PCPP verifier V^{comp} will use the inner PCPP verifier V^{in} to simulate the outer PCPP verifier V^{out}. This means that the total number of queries and parity-check functions is at most the query complexity of the inner PCPP verifier. Moreover, the decision predicate of V^{comp} (after fixing the random seed) is the decision predicate of V^{in}, where the input bits of the decision circuit of V^{comp} are the parity-check bits and the answers to the queries. For instance, if the decision predicate of V^{in} given $\mathsf{seed}^{\mathsf{in}}$ is an OR of the answers or their negations, then the decision predicate of V^{comp} given $\mathsf{seed} = (\mathsf{seed}^{\mathsf{in}}, \mathsf{seed}^{\mathsf{out}})$ is also the same OR of its input bits (i.e. the answers to the queries and the parity-check bits).

6.4 Soundness Amplification Preserving Smoothness and Rectangularity

Now we use the technique from [RSW22, Section 7.1.3] to boost the soundness error of a rectangular PCPP. In addition to their original analysis, we need to verify that their expander-walk construction preserves smoothness.

Lemma 6.12 ([VW18, RSW22])**.** *For every $\lambda \in (0,1)$, there is some $d = \mathrm{poly}(\lambda^{-1})$ such that the following holds. For every n, there is an expander graph $G_n = (V_n, E_n)$ with second largest eigenvalue at most λ, where $V_n := \{0,1\}^n$. Moreover, 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 for every $x \in V_n$, the d neighbours of x are $C_1(x), C_2(x), \ldots, C_d(x)$.*

Verifier	V^{old}	V^{new}
Soundness error	s	μ
Proximity parameter	δ	δ
Row randomness	r_{row}	r_{row}
Column randomness	r_{col}	r_{col}
Shared randomness	r_{shared}	$r_{\mathsf{shared}} + O((1-s)^{-2}\log(\mu^{-1})\log((1-s)^{-1}))$
Proof matrix height	H_{proof}	H_{proof}
Proof matrix width	W_{proof}	W_{proof}
Query complexity	q	$O(q \cdot (1-s)^{-2} \cdot \log(\mu^{-1}))$
Parity check complexity	p	$O(p \cdot (1-s)^{-2} \cdot \log(\mu^{-1}))$
Decision complexity	d	$O(d \cdot (1-s)^{-2}\log(\mu^{-1}) + \mathrm{poly}(r_{\mathsf{shared}}, r_{\mathsf{row}}, r_{\mathsf{col}}))$

Table 6: The parameters of the soundness amplification ($O(\cdot)$ hides absolute constants).

Lemma 6.13 (Expander Walk, [AB09, Theorem 21.12])**.** *Let $G = (V, E)$ be a d-regular graph with second largest eigenvalue λ. For every $S \subseteq V$ such that $|S| \leq \beta \cdot |V|$ for some $\beta \in (0,1)$, let $(X_1, X_2, \ldots, X_\ell)$ be a random walk in G with random starting point, then*

$$\Pr\left[\forall i \in [\ell], X_i \in S\right] \leq \left((1-\lambda)\sqrt{\beta} + \lambda\right)^{\ell-1}.$$

Lemma 6.14 (Expander Chernoff Bound, [Vad12, Theorem 4.22])**.** *Let $G = (V, E)$ be a d-regular graph with second largest eigenvalue λ, $B \subseteq V$ be a set of size $|B| = \beta|V|$. Let $X_1, X_2, \ldots, X_\ell$ be random variables denoting a length-ℓ random walk from a random starting point. For every $i \in [\ell]$, we define $B_i = 1$ if $X_i \in B$ and $B_i = 0$ otherwise. Then:*

$$\Pr\left[\left|\frac{1}{\ell}\sum_{i=1}^{\ell} B_i - \beta\right| \geq 2\lambda\right] < 2\exp\left(-\Omega(\lambda^2\ell)\right).$$

Theorem 6.15. *Suppose that L has a rectangular PCPP verifier V^{old} (resp. a rectangular PCPP verifier V^{old} with ROP), where the parameters are specified in Table 6. Then for every $\mu \in (0,1)$, L has a rectangular PCPP verifier V^{new} (resp. a rectangular PCPP verifier with ROP), whose parameters are specified in Table 6.*

Moreover, if V^{old} is smooth, then V^{new} is also smooth; if V^{old} has robust soundness (instead of soundness) s with robustness parameter ρ, then V^{old} has robustness soundness μ with robustness parameter $(1-s)\rho/3$.

Proof. Let $\varepsilon := 1 - s$, $\lambda := \varepsilon/3$, and $r := r_{\mathsf{row}} + r_{\mathsf{col}} + r_{\mathsf{shared}}$. We construct the following d-regular expander graphs with second largest eigenvalue λ by Lemma 6.12: $G_{\mathsf{row}} = (V_{\mathsf{row}}, E_{\mathsf{row}})$ with $V_{\mathsf{row}} := \{0,1\}^{r_{\mathsf{row}}}$, $G_{\mathsf{col}} = (V_{\mathsf{col}}, E_{\mathsf{col}})$ with $V_{\mathsf{col}} := \{0,1\}^{r_{\mathsf{col}}}$, and $G_{\mathsf{shared}} = (V_{\mathsf{shared}}, E_{\mathsf{shared}})$ with $V_{\mathsf{shared}} := \{0,1\}^{r_{\mathsf{shared}}}$. Let $G = (V, E)$ be the tensor product of these expanders:

$$V := V_{\mathsf{row}} \times V_{\mathsf{col}} \times V_{\mathsf{shared}} = \{0,1\}^{r_{\mathsf{row}}} \times \{0,1\}^{r_{\mathsf{col}}} \times \{0,1\}^{r_{\mathsf{shared}}};$$
$$E := \{((u,v,w),(u',v',w')) : (u,u') \in E_{\mathsf{row}}, (v,v') \in E_{\mathsf{col}}, (w,w') \in E_{\mathsf{shared}}\}.$$

Note that G is a d^3-regular graph with second largest eigenvalue λ (see [AB09, Lemma 21.17]).

The Construction of V^{new}. The new verifier has the same proof matrix, row randomness, and column randomness as the old verifier V^{old}. The shared random seed of the new verifier V^{new} consists of the shared random seed $\mathsf{seed.shared}$ of V^{old} and $\mathsf{seed.walk}$, which is used to sample a random walk in G of length $\ell := O(\lambda^{-2}\log(\mu^{-1}))$. Concretely:

- The random seed $\mathsf{seed.walk}$ will be used to sample $\sigma_1, \sigma_2, \ldots, \sigma_{3(\ell-1)} \in [d]$. We can see that

$$|\mathsf{seed.walk}| = O(\ell \cdot \log d) = O(\lambda^{-2}\log(\mu^{-1})\log(\lambda^{-1})).$$

- Let $u_1 := \mathsf{seed.row}$, $v_1 := \mathsf{seed.col}$, and $w_1 := \mathsf{seed.shared}$. We use $\sigma_1, \sigma_2, \ldots, \sigma_{\ell-1}$ to specify a length-ℓ random walk $(u_1, u_2, \ldots, u_\ell)$ in G_{row}. In particular, let $C_1, C_2, \ldots, C_d$ be the projections in Lemma 6.12 for G_{row}. For every $j \in \{1, 2, \ldots, \ell-1\}$, we define $u_{j+1} := C_{\sigma_j}(u_j)$. Similarly, we can use the remaining $2(\ell-1)$ bits to specify a random walk $(v_1, v_2, \ldots, v_\ell)$ in G_{col} and a random walk $(w_1, w_2, \ldots, w_\ell)$ in G_{shared}.

The verifier V^{old} will run the verifier V^{new} for ℓ times with the seeds:

$$(u_1, v_1, w_1), (u_2, v_2, w_2), \ldots, (u_\ell, v_\ell, w_\ell),$$

and will accept the proof if V^{old} accepts given all these ℓ seeds. Since $\mathsf{seed.walk}$ is treated as the shared randomness of V^{new} and G is obtained from the tensor product of G_{row}, G_{col}, and G_{shared}, it is easy to see that V^{new} is still a rectangular PCPP verifier. The query complexity (and parity-check complexity when V^{old} has ROP) increases by an $\ell = O(\varepsilon^{-2}\log(\mu^{-1}))$ multiplicative factor.

Smoothness. Let $\mathsf{idx} \in [H_{\mathsf{proof}} \cdot W_{\mathsf{proof}}]$ be an index in the proof. Assume that $V^{\mathsf{new}}(\mathsf{seed}, i)$ (resp. $V^{\mathsf{old}}(\mathsf{seed}, i)$) denotes the index in the proof probed by V^{new} (resp. V^{old}) for the i-th query. We can see that

$$\begin{aligned}\gamma &:= \Pr_{\mathsf{seed},\mathsf{seed.walk},i\in[q\ell]}[V^{\mathsf{new}}(\mathsf{seed} \circ \mathsf{seed.walk}, i) = \mathsf{idx}] \\ &= \mathop{\mathbb{E}}_{j\in[\ell]}\left[\Pr_{\mathsf{seed},\mathsf{seed.walk},i\in[q]}[V^{\mathsf{new}}(\mathsf{seed} \circ \mathsf{seed.walk}, (j-1)\ell + i) = \mathsf{idx}]\right].\end{aligned}$$

Fix a $j \in [\ell]$. By the definition of V^{new}, we know that $V^{\mathsf{new}}(\mathsf{seed} \circ \mathsf{seed.walk}, (j-1)\ell + i)$ will work as follows: Let $(u, v, w) := (\mathsf{seed.row}, \mathsf{seed.col}, \mathsf{seed.shared})$, and $\sigma_1, \sigma_2, \ldots, \sigma_{3\ell}$ be defined as above; V^{new} will choose the j-th node in the random walk on G seeded by $\mathsf{seed.walk}$ starting from $(u_1 := u, v_1 := v, w_1 := w)$ as the seed for V^{old}, and probe the proof according to the i-th query of V^{old}. Since the expander graph is regular, each $\mathsf{seed} \in \{0,1\}^r$ is equally likely to be selected from a random walk with a random starting point. Hence

$$\begin{aligned}&\Pr_{\mathsf{seed},\mathsf{seed.walk},i\in[q]}[V^{\mathsf{new}}(\mathsf{seed} \circ \mathsf{seed.walk}, (j-1)\ell + i) = \mathsf{idx}] \\ &= \Pr_{\mathsf{seed},i\in[q]}\left[V^{\mathsf{old}}(\mathsf{seed}, i) = \mathsf{idx}\right] \\ &= \frac{1}{H_{\mathsf{proof}} \cdot W_{\mathsf{proof}}}.\end{aligned}$$

This means that $\gamma = 1/(H_{\mathsf{proof}} \cdot W_{\mathsf{proof}})$, i.e., every bit in the proof is equally likely to be probed.

Soundness. Assume that $x \in \{0,1\}^n$ is δ-far from being in L and Π be an arbitrary proof. We say a node (u, v, w) in the expander graph $G = (V, E)$ to be *bad* if V^{old} accepts (x, Π) with the random seed $\mathsf{seed.row} := u$, $\mathsf{seed.col} := v$, and $\mathsf{seed.shared} := w$. Let B be the set of all bad nodes, then $|B| \leq s \cdot |V|$. Note that the new verifier accepts (x, Π) if and only if a length-ℓ random walk on G from a random starting point only accesses bad nodes. By Lemma 6.13, we can see that

$$\Pr\left[V^{\mathsf{new}} \text{ accepts } (x,\Pi)\right] \leq \left((1-\lambda)\sqrt{s}+\lambda\right)^{\ell-1} \leq \left(1-\frac{\varepsilon}{5}\right)^{\ell-1} \leq \exp\left(-\frac{\varepsilon(\ell-1)}{5}\right) \leq \mu,$$

when $\ell \geq 10 \cdot \varepsilon^{-1} \ln(\mu^{-1})$.

Robust Soundness. Assume that the original PCPP verifier V^{old} has robust soundness s with robustness parameter ρ (instead of only soundness s), we need to show that V^{new} has robustness soundness μ. Let $x \in \{0,1\}^n$ be δ-far from L and Π be an arbitrary proof. We say a node (u, v, w) in the expander graph $G = (V, E)$ is *bad* if given the randomness $\mathsf{seed} = (\mathsf{seed.row}, \mathsf{seed.col}, \mathsf{seed.shared}) := (u, v, w)$, the fraction of bits read by the old PCPP verifier that we need to change to make V^{old} accepts (x, Π), denoted by $\hat{\delta}(\mathsf{seed})$, is at most ρ.

Let B be the set of bad nodes and $X_1, X_2, \ldots, X_\ell$ be the random variables denoting a random walk from a random starting point (equivalently, denoting the randomness V^{new} used to simulate V^{old}). By the robustness soundness of V^{old}, we know that $|B| \leq s \cdot |V|$. Let $B_i = 1$ when $X_i \in B$ and 0 otherwise. By Lemma 6.14, we can see that

$$\Pr\left[\frac{1}{\ell}\sum_{i=1}^{\ell} B_i \geq s + 2\lambda\right] \leq 2\exp(-\Omega(\lambda^2\ell)) \leq \mu$$

when $\ell = O(\lambda^{-2}\log(\mu^{-1}))$. As a result, with probability as least $1-\mu$, the fraction of bits read by V^{new} that we need to change to make V^{new} accepts (x, Π) is at least

$$(1-(s+2\lambda))\rho \geq \varepsilon\rho/3.$$

This satisfies the requirement of robust soundness μ with robustness parameter $\varepsilon\rho/3$. □

6.5 Final Construction

Theorem 6.16 ([Mie09,RSW22]). *Let L be a pair language in $\mathsf{NTIME}[T(n)]$ for some non-decreasing function $T : \mathbb{Z}^+ \to \mathbb{Z}^+$. For all constants $s, \delta > 0$, L has a PCPP verifier with randomness complexity $\log T(n) + O(\log\log T(n))$, soundness error s, proximity parameter δ, query complexity $O(1)$, and decision complexity $\mathrm{polylog}(T(n))$.*

Theorem 2.14 (Smooth and Rectangular PCPP). *For all constants $\delta \in (0,1)$ and $s \in (0,1)$, there is a constant $q \geq 1$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \leq m(n) \leq (\log T(n))^{0.1}$, $n \leq T(n) \leq 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \leq \log T(n)$, and $w_{\mathsf{input}}(n) \leq \log n$. Then there are good functions $h_{\mathsf{proof}}(n)$ and $h_{\mathsf{input}}(n)$ satisfying*

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

such that the following holds.

Soundness error	s
Proximity parameter	δ
Row randomness	$r_{\mathsf{row}} := h_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$r_{\mathsf{col}} := w_{\mathsf{proof}} - (5/m)\log T(n)$
Shared randomness	$r_{\mathsf{shared}} := (10/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	q
Parity check complexity	q
Decision complexity	$\mathrm{poly}(T(n)^{1/m})$

Table 7: Parameters of the PCPP constructed in Theorem 2.14.

Suppose that $w_{\mathsf{proof}}, h_{\mathsf{proof}} \geq (5/m)\log T(n)$*, and that for some absolute constant* $C \geq 1$*,*

$$\frac{w_{\mathsf{input}}(n)}{w_{\mathsf{proof}}(n)}, \frac{h_{\mathsf{input}}(n)}{h_{\mathsf{proof}}(n)} \leq 1 - \frac{Cm^2 \log\log T(n)}{\log T(n)}.$$

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 smooth and rectangular PCP of proximity with an* $H_{\mathsf{input}}(n) \times W_{\mathsf{input}}(n)$ *input matrix and an* $H_{\mathsf{proof}}(n) \times W_{\mathsf{proof}}(n)$ *proof matrix, whose other parameters are specified in Table 2.*

Proof. The high-level roadmap of the proof is as follows.

1. By Theorem 6.8, we can obtain a robust and rectangular PCPP verifier V^{out} with RNL property for $t_{\mathsf{RNL}} = \mathrm{poly}(\log T(n), m^m)$ and query complexity $T(n)^{1/m} \cdot \mathrm{polylog}(T(n))$.
2. Let V^{in} be a PCPP verifier for Circuit-Eval$^{\perp}$ with constant query complexity. We compose V^{out} and V^{in} by Theorem 6.10 to obtain a rectangular PCPP verifier V^{comp} with RNL property.
3. We smoothen V^{comp} by Theorem 6.3 to obtain a smooth and rectangular PCPP V^{smth} with constant query complexity, whose soundness error is some constant $s^{\mathsf{smth}} \in (0,1)$.
4. By Theorem 6.15, we reduce the soundness error to s while still maintaining the query complexity to be a (larger) constant.

Robust and Rectangular PCPP. Let $\delta \in (0,1)$ and $s \in (0,1)$ be some constants; q be a large constant to be determined that only depends on δ and s; C be a large constant; $m = m(n)$, $T(n)$, $w_{\mathsf{input}}(n)$, $h_{\mathsf{input}}(n)$, $w_{\mathsf{proof}}(n)$, and $h_{\mathsf{proof}}(n)$ be defined as above. Let $w^{\mathsf{out}}_{\mathsf{proof}}(n) := w_{\mathsf{proof}}(n) - O(\log\log T(n) + m\log m)$ where the concrete value will be determined later. We will set $h_{\mathsf{proof}}(n) = h^{\mathsf{out}}_{\mathsf{proof}}(n) + O(\log\log T(n) + m\log m)$ for some good function $h^{\mathsf{out}}_{\mathsf{proof}}(n)$ (which is actually the proof height parameter of the outer PCPP). We check the technical conditions of Theorem 6.8 holds; in particular we need to ensure that

Claim 6.17. *For some constant* C' *that could be made large enough (depending on* C*),*

$$\frac{w_{\mathsf{input}}}{w^{\mathsf{out}}_{\mathsf{proof}}}, \frac{h_{\mathsf{input}}}{h^{\mathsf{out}}_{\mathsf{proof}}} \leq 1 - \frac{C'm\log\log T(n)}{\log T(n)}.$$

Proof. Since $w_{\mathsf{proof}} \geq (5/m)\log T(n)$, and $w_{\mathsf{proof}} - w^{\mathsf{out}}_{\mathsf{proof}} \leq \alpha_1(\log\log T(n) + m\log m)$ for some constant α_1, it follows that

$$\begin{aligned}\frac{w_{\mathsf{input}}}{w^{\mathsf{out}}_{\mathsf{proof}}} &\leq \frac{w_{\mathsf{input}}}{w_{\mathsf{proof}}}\left(1 + \frac{\alpha_1(\log\log T(n) + m\log m)}{w_{\mathsf{proof}} - \alpha_1(\log\log T(n) + m\log m)}\right)\\ &\leq 1 - \frac{Cm^2\log\log T(n)}{\log T(n)} + \frac{\Theta(m)\cdot(\log\log T(n) + m\log m)}{\log T(n)}\\ &\leq 1 - \frac{C'm\log\log T(n)}{\log T(n)}.\end{aligned}$$

The same argument works for $\frac{h_{\mathsf{input}}}{h^{\mathsf{out}}_{\mathsf{proof}}}$. ◇

By Theorem 6.8, we can construct a robust and rectangular PCPP verifier V^{out} for L with RNL property and other parameters as follows:

- Proximity parameter $\delta^{\mathsf{out}} := \delta$.
- Robust soundness error $s^{\mathsf{out}} := 1 - \rho^{\mathsf{out}}$ with robustness parameter ρ^{out}, where $\rho^{\mathsf{out}} \in (0,1)$ is some constant depending on δ.
- Proof matrix size $H^{\mathsf{out}}_{\mathsf{proof}}(n) \times W^{\mathsf{out}}_{\mathsf{proof}}(n)$, where $H^{\mathsf{out}}_{\mathsf{proof}} = 2^{h^{\mathsf{out}}_{\mathsf{proof}}}$, $W^{\mathsf{out}}_{\mathsf{proof}} = 2^{w^{\mathsf{out}}_{\mathsf{proof}}}$. The proof height parameter $h^{\mathsf{out}}_{\mathsf{proof}}$, which is given by Theorem 6.8, satisfies

$$h^{\mathsf{out}}_{\mathsf{proof}} = \log T(n) + \Theta(m\log\log T(n)) - w^{\mathsf{out}}_{\mathsf{proof}}(n).$$

- Row randomness complexity $r^{\mathsf{out}}_{\mathsf{row}} = h^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Column randomness complexity $r^{\mathsf{out}}_{\mathsf{col}} = w^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Shared randomness complexity $r^{\mathsf{out}}_{\mathsf{shared}} = (7/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Query complexity $q^{\mathsf{out}}(n) = T(n)^{1/m}\cdot\mathrm{polylog}(T(n))$.
- Decision complexity $d^{\mathsf{out}}(n) = T(n)^{1/m}\cdot\mathrm{polylog}(T(n))$.
- RNL time complexity $t^{\mathsf{out}}_{\mathsf{RNL}}(n) = \mathrm{poly}(\log T(n), m^m)$.

Reducing the Query Complexity. By Theorem 6.16, we can construct a PCPP verifier V^{in} for Circuit-Eval$^{\perp}$ with input length $d^{\mathsf{out}}(n)$ and other parameters specified as follows.

- Randomness complexity $r^{\mathsf{in}}(n) = \log d^{\mathsf{out}}(n) + O(\log\log d^{\mathsf{out}}(n)) = \frac{1}{m}\log T(n) + O(\log\log T(n))$.
- Soundness error $s^{\mathsf{in}} := \rho^{\mathsf{out}}/2$.
- Proximity parameter $\delta^{\mathsf{in}} := \rho^{\mathsf{out}}/2$.
- Query complexity $q^{\mathsf{in}} = O(1)$ is a constant depends on s^{in} and δ^{in}, which further means that it only depends on δ.
- Decision complexity $d^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \mathrm{polylog}(T(n))$.

Without loss of generality, we assume that the proof length $\ell^{\mathsf{in}}(n) = 2^{r^{\mathsf{in}}(n)}$.

We now construct V^{comp} by composing V^{out} and V^{in} by Theorem 6.10. We firstly check the requirements of the composition theorem.

- $q^{\mathsf{in}} = O(1)$, $\rho^{\mathsf{out}} \geq \delta^{\mathsf{in}} = \rho^{\mathsf{out}}/2$, $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$.
- $\log W^{\mathsf{out}}_{\mathsf{proof}} = r^{\mathsf{out}}_{\mathsf{col}} + (4/m)\log T(n) \geq r^{\mathsf{out}}_{\mathsf{col}}$.
- $\log W^{\mathsf{out}}_{\mathsf{proof}} = r^{\mathsf{out}}_{\mathsf{col}} + (4/m)\log T(n) \leq r^{\mathsf{out}}_{\mathsf{col}} + r^{\mathsf{out}}_{\mathsf{shared}}$.

Hence we can obtain a rectangular PCPP V^{comp} with ROP that has RNL property. The parameters of the composed PCPP are as follows.

- Soundness error $s^{\mathsf{comp}} := 1 - (1 - s^{\mathsf{out}}) \cdot (1 - s^{\mathsf{in}}) < 1$ that only depends on δ.
- Proximity parameter $\delta^{\mathsf{comp}} := \delta^{\mathsf{out}} = \delta$.
- Row randomness complexity $r^{\mathsf{comp}}_{\mathsf{row}} = r^{\mathsf{out}}_{\mathsf{row}} = h^{\mathsf{out}}_{\mathsf{proof}} - (4/m) \log T(n)$.
- Column randomness complexity $r^{\mathsf{comp}}_{\mathsf{col}} = r^{\mathsf{out}}_{\mathsf{col}} = w^{\mathsf{out}}_{\mathsf{proof}} - (4/m) \log T(n)$.
- Shared randomness complexity $r^{\mathsf{comp}}_{\mathsf{shared}} = r^{\mathsf{out}}_{\mathsf{shared}} + r^{\mathsf{in}} = (7/m) \log T(n) + O(\log \log T(n) + m \log m) + (1/m) \log T(n) + O(\log \log T(n)) = (8/m) \log T(n) + O(\log \log T(n) + m \log m)$.
- Proof matrix height $H^{\mathsf{comp}}_{\mathsf{proof}} = H^{\mathsf{out}}_{\mathsf{proof}} + 2^{r^{\mathsf{out}} + r^{\mathsf{in}}} / W^{\mathsf{out}}_{\mathsf{proof}}$.
- Proof matrix width $W^{\mathsf{comp}}_{\mathsf{proof}} = W^{\mathsf{out}}_{\mathsf{proof}}$.
- Query complexity $q^{\mathsf{comp}} = q^{\mathsf{in}} = O(1)$ that only depends on δ.
- ROP parity check complexity $p^{\mathsf{comp}} = q^{\mathsf{in}} = O(1)$.
- Decision complexity $d^{\mathsf{comp}}(n) = d^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \mathrm{polylog}(T(n))$.
- RNL time complexity $t^{\mathsf{comp}}_{\mathsf{RNL}}(n) = \mathrm{poly}(t^{\mathsf{out}}_{\mathsf{RNL}}(n), \ell^{\mathsf{in}}, q^{\mathsf{in}}, d^{\mathsf{in}}) = \mathrm{poly}\big(T(n)^{1/m}\big)$, where $\mathrm{poly}(\cdot)$ hides some absolute constant on the exponent. Note that $t^{\mathsf{out}}_{\mathsf{RNL}}(n) = \mathrm{poly}(\log T(n), m^m) \leq \mathrm{poly}(T(n)^{1/m})$, since $m \leq (\log T(n))^{0.1}$.

Smoothening via RNL. Now we apply Theorem 6.3 to obtain a smooth and rectangular PCPP V^{smth} with $\mu := (1 - s^{\mathsf{comp}})/2$ and other parameters as follows.

- Soundness error $s^{\mathsf{smth}} := s^{\mathsf{comp}} + \mu < 1$ that only depends on δ.
- Proximity parameter $\delta^{\mathsf{smth}} := \delta^{\mathsf{comp}} = \delta$.
- Row randomness complexity $r^{\mathsf{smth}}_{\mathsf{row}} := r^{\mathsf{comp}}_{\mathsf{row}} = h^{\mathsf{out}}_{\mathsf{proof}} - (4/m) \log T(n)$.
- Column randomness complexity $r^{\mathsf{smth}}_{\mathsf{col}} := r^{\mathsf{comp}}_{\mathsf{col}} = w^{\mathsf{out}}_{\mathsf{proof}} - (4/m) \log T(n)$.
- Shared randomness complexity $r^{\mathsf{smth}}_{\mathsf{shared}} = r^{\mathsf{comp}}_{\mathsf{shared}} = (8/m) \log T(n) + O(\log \log T(n) + m \log m)$.
- Proof matrix width $W^{\mathsf{smth}}_{\mathsf{proof}} = 2^{r^{\mathsf{comp}}_{\mathsf{col}} + r^{\mathsf{comp}}_{\mathsf{shared}}/2} = 2^{w^{\mathsf{out}}_{\mathsf{proof}}} \cdot \mathrm{poly}(\log T(n), m^m)$. Note that here we set $w^{\mathsf{out}}_{\mathsf{proof}}$ carefully so that $r^{\mathsf{comp}}_{\mathsf{col}} + r^{\mathsf{comp}}_{\mathsf{shared}}/2 = w^{\mathsf{out}}_{\mathsf{proof}} + O(\log \log T(n) + m \log m) = w_{\mathsf{proof}}$. This means that the proof matrix width is exactly $2^{w_{\mathsf{proof}}}$.
- Proof matrix height $H^{\mathsf{smth}}_{\mathsf{proof}} = q^{\mathsf{comp}} \cdot 2^{r^{\mathsf{comp}}_{\mathsf{row}} + r^{\mathsf{comp}}_{\mathsf{shared}}/2} = 2^{h^{\mathsf{out}}_{\mathsf{proof}}} \cdot \mathrm{poly}(\log T(n), m^m)$. Since $q^{\mathsf{comp}} = O(1)$, we can add $O(1)$ dummy queries to the composed PCPP V^{comp} so that q^{comp} becomes a power of two. We then set

$$\begin{aligned} h_{\mathsf{proof}} &= \log H^{\mathsf{smth}}_{\mathsf{proof}} \\ &= h^{\mathsf{out}}_{\mathsf{proof}} + O(\log \log T + m \log m) \\ &= \log T(n) + \Theta(m \log \log T(n)) + O(\log \log T + m \log m) \\ &\quad - (w_{\mathsf{proof}}(n) - O(\log \log T + m \log m)) \\ &= \log T(n) + \Theta(m \log \log T(n)) - w_{\mathsf{proof}}(n). \end{aligned}$$

- Query complexity $q^{\mathsf{smth}} = \mathrm{poly}(q^{\mathsf{comp}}/\mu) = O(1)$ that only depends on δ.
- ROP parity check complexity $p^{\mathsf{smth}} = p^{\mathsf{comp}} = O(1)$.
- Decision complexity $d^{\mathsf{smth}}(n) = \mathrm{poly}(d^{\mathsf{comp}}(n), q^{\mathsf{comp}}/\mu, t^{\mathsf{comp}}_{\mathsf{RNL}}(n)) = \mathrm{poly}\big(T(n)^{1/m}\big)$, where $\mathrm{poly}(\cdot)$ hides some absolute constant on the exponent.

Amplifying the Soundness Error. Finally, we boost the soundness error of V^{smth} to be s by Theorem 6.15, to obtain a smooth and rectangular PCPP with parameters specified as follows.

- Soundness error s.
- Proximity parameter δ.
- Row randomness complexity $h^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n) \geq h_{\mathsf{proof}} - (5/m)\log T(n)$.
- Column randomness complexity $w^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n) \geq w_{\mathsf{proof}} - (5/m)\log T(n)$.
- Shared randomness complexity $(8/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Proof matrix height $2^{h_{\mathsf{proof}}}$ and proof matrix width $2^{w_{\mathsf{proof}}}$.
- Query complexity $q = \mathrm{poly}(q^{\mathsf{smth}}) = O(1)$ that depends on δ and s.
- ROP parity check complexity $\mathrm{poly}(p^{\mathsf{smth}}) = O(1)$ that depends on δ and s.
- Decision complexity $\mathrm{poly}(d^{\mathsf{smth}}(n)) = \mathrm{poly}(T(n)^{1/m})$.

We can move some bits from the row and column randomness to the shared randomness, so that the row and column randomness complexity become exactly $h_{\mathsf{proof}} - (5/m)\log T(n)$ and $w_{\mathsf{proof}} - (5/m)\log T(n)$, respectively, and the shared randomness complexity becomes $(10/m)\log T(n) + O(\log\log T(n) + m\log m)$. This completes the construction. □

7 Construction of Rectangular PCPPs with Low Query Complexity

Recall that in our framework of solving range avoidance and hard partial truth table, the query complexity of the PCPPs will affect the circuit class for which we need to construct satisfying-pair algorithms. In this section, we construct a rectangular (but not necessarily smooth) PCPP with query complexity 3. We further construct a 2-query PCPP with a constant gap between the completeness and soundness parameters (instead of having perfect completeness).

7.1 A 3-Query PCPP for Circuit-Eval$^{\perp}$

Theorem 7.1 ([CW19b], Lemma 24)**.** *For every constant $\delta > 0$, there is a constant $s \in (0,1)$ and a PCP of proximity for* Circuit-Eval *with proximity δ, soundness error s, randomness complexity $O(\log n)$, query complexity $q = 3$, and decision complexity* $\mathrm{polylog}(n)$*. Moreover, the decision predicate is an* OR *of the* 3 *answers to the queries or their negations.*

We need the following standard composition theorem for PCP of Proximity from [BGH+06] to construct 3-query PCPPs for any pair language in $\mathsf{NTIME}[T(n)]$.

Theorem 7.2 ([BGH+06])**.** *Let $r^{\mathsf{out}}, r^{\mathsf{in}}, d^{\mathsf{out}}, d^{\mathsf{in}}, q^{\mathsf{in}} : \mathbb{N} \to \mathbb{N}$ and $\varepsilon^{\mathsf{out}}, \varepsilon^{\mathsf{in}}, \rho^{\mathsf{out}}, \delta^{\mathsf{in}}, \delta^{\mathsf{out}} : \mathbb{N} \to [0,1]$. Suppose that:*

- *Language L has a robust PCPP verifier V^{out} with randomness complexity $r^{\mathsf{out}}(n)$, decision complexity $d^{\mathsf{out}}(n)$, robust soundness error $1 - \varepsilon^{\mathsf{out}}(n)$, robustness parameter $\rho^{\mathsf{out}}(n)$, and proximity parameter $\delta^{\mathsf{out}}(n)$.*
- Circuit-Eval *has a PCPP verifier V^{in} with randomness complexity $r^{\mathsf{in}}(n)$, query complexity $q^{\mathsf{in}}(n)$, decision complexity $d^{\mathsf{in}}(n)$, soundness error $1-\varepsilon^{\mathsf{in}}(n)$, and proximity parameter $\delta^{\mathsf{in}}(n)$.*
- *$\delta^{\mathsf{in}}(d^{\mathsf{out}}(n)) \leq \rho^{\mathsf{out}}(n)$ for every n.*

Then L has a PCPP Verifier V^{comp} with randomness complexity $r^{\mathsf{out}}(n) + r^{\mathsf{in}}(d^{\mathsf{out}}(n))$, query complexity $q^{\mathsf{in}}(d^{\mathsf{out}}(n))$, decision complexity $d^{\mathsf{in}}(d^{\mathsf{out}}(n))$, soundness error $1 - \varepsilon^{\mathsf{out}}(n) \cdot \varepsilon^{\mathsf{in}}(d^{\mathsf{out}}(n))$, and proximity parameter $\delta^{\mathsf{out}}(n)$.

Theorem 7.3. *Let L be a pair language in $\mathsf{NTIME}[T(n)]$ for some non-decreasing function $T : \mathbb{Z}^+ \to \mathbb{Z}^+$. For every constant δ, there is a constant $s \in (0,1)$ and a PCP of proximity for L with randomness complexity $\log T(n) + O(\log\log T(n))$, decision complexity $\mathrm{poly}(\log\log T(n))$, soundness error s, proximity parameter δ, and query complexity $q = 3$.*

Proof. Let L be a pair language in $\mathsf{NTIME}[T(n)]$ and $\delta > 0$. We will compose the following two PCPP verifiers with Theorem 7.2:

- By Theorem 7.1, for every $\delta^{\mathsf{in}} > 0$, there is a constant $s^{\mathsf{in}} \in (0,1)$ and a PCPP verifier $V^{\mathsf{in}}_{\delta^{\mathsf{in}}}$ for Circuit-Eval with randomness complexity $r^{\mathsf{in}} = O(\log n)$, soundness error s^{in}, proximity parameter δ^{in}, query complexity $q^{\mathsf{in}} = 3$, and decision complexity $d^{\mathsf{in}} = \mathrm{polylog}(n)$.
- By Theorem 6.16, for all constants $\delta^{\mathsf{out}}, s^{\mathsf{out}} > 0$, there is a constant q^{out} and a PCPP V^{out} with randomness complexity $r^{\mathsf{out}} = \log T(n) + O(\log\log T(n))$, soundness error s^{out}, proximity parameter δ^{out}, query complexity q^{out}, and decision complexity $d^{\mathsf{out}} = \mathrm{polylog}(T(n))$. Since $q^{\mathsf{out}} = O(1)$, V^{out} is trivially a robust PCPP with robustness parameter $\rho^{\mathsf{out}} = 1/q^{\mathsf{out}}$.

Fix $\delta^{\mathsf{out}} = \delta$, $s^{\mathsf{out}} = 0.5$, $\delta^{\mathsf{in}} = 1/(2q^{\mathsf{out}})$, and $s = 1 - 0.5 \cdot (1 - s^{\mathsf{in}})$. It is clear that $\delta^{\mathsf{in}}(d^{\mathsf{out}}(n)) \le \rho^{\mathsf{out}}(n)$. By Theorem 7.2, we can obtain a PCPP verifier V^{comp} for L with the following parameters:

- Randomness complexity $r^{\mathsf{out}} + r^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \log T(n) + O(\log\log T(n))$.
- Decision complexity $d^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \mathrm{poly}(\log\log T(n))$.
- Soundness error $1 - (1 - s^{\mathsf{out}}(n)) \cdot (1 - s^{\mathsf{in}}(d^{\mathsf{out}}(n))) = s$.
- Proximity parameter $\delta^{\mathsf{out}}(n) = \delta$.
- Query complexity $q^{\mathsf{in}}(d^{\mathsf{out}}(n)) = 3$.

This satisfies our requirements. □

7.2 A 3-Query Rectangular PCPP

Now we construct a 3-query rectangular PCPP by composing the PCPP constructions in Theorem 6.8 and Theorem 7.3 using the composition theorem (see Theorem 6.10).

Theorem 7.4 (3-Query Rectangular PCPP)**.** *For every constant $\delta \in (0,1)$, there is a constant $s \in (0,1)$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \le m \le (\log T(n))^{0.1}$, $n \le T(n) \le 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \le \log T(n)$, and $w_{\mathsf{input}}(n) \le \log n$. Then there are good functions $h_{\mathsf{proof}}(n)$ and $h_{\mathsf{input}}(n)$ satisfying*

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

such that the following holds.

Suppose that $w_{\mathsf{proof}}, h_{\mathsf{proof}} \ge (5/m) \log T(n)$, and that for some absolute constant $C \ge 1$,

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

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 8.

Moreover, the total number of queries and parity-check bits is at most 3; and for every $\mathsf{seed.shared}$, the decision predicate $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ of the rectangular PCPP verifier is an OR of its 3 input bits or their negations, where each input is either a query answer or a parity-check bit.

Soundness error	s
Proximity parameter	δ
Row randomness	$h_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (5/m)\log T(n)$
Shared randomness	$(10/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	3
Parity check complexity	
Decision complexity	$\mathrm{poly}(\log\log T(n))$

Table 8: Parameters of the PCPP constructed in Theorem 7.4.

Proof. Let $L \in \mathsf{NTIME}[T(n)]$ and $m \geq 1, \delta > 0$ be constants; $T(n)$, $w_{\mathsf{proof}}(n)$, $h_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$, $h_{\mathsf{input}}(n)$, and C be defined as above. In one sentence, we compose the robust and rectangular PCPP verifier (Theorem 6.8) with the 3-query PCPP verifier (Theorem 7.3) using the composition theorem (Theorem 6.10).

Outer PCPP. Let $w^{\mathsf{out}}_{\mathsf{proof}}(n) := w_{\mathsf{proof}}(n)$. By Theorem 6.8, we can construct a robust and rectangular PCPP verifier V^{out} for L with parameters as follows:

- Robust soundness error $s^{\mathsf{out}} \in (0,1)$ with robustness parameter $\rho^{\mathsf{out}} := 1 - s^{\mathsf{out}}$.
- Proximity parameter $\delta^{\mathsf{out}} := \delta$.
- Proof matrix size $H^{\mathsf{out}}_{\mathsf{proof}}(n) \times W^{\mathsf{out}}_{\mathsf{proof}}(n)$, where $H^{\mathsf{out}}_{\mathsf{proof}} = 2^{h^{\mathsf{out}}_{\mathsf{proof}}}$, $W^{\mathsf{out}}_{\mathsf{proof}} = 2^{w_{\mathsf{proof}}}$, and $h^{\mathsf{out}}_{\mathsf{proof}} = \log T(n) + \Theta(m\log\log T(n)) - w_{\mathsf{proof}}(n)$.
- Row randomness complexity $r^{\mathsf{out}}_{\mathsf{row}} = h^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$.
- Column randomness complexity $r^{\mathsf{out}}_{\mathsf{col}} = w_{\mathsf{proof}} - (4/m)\log T(n)$.
- Shared randomness complexity $r^{\mathsf{out}}_{\mathsf{shared}} = (7/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Query complexity $q^{\mathsf{out}}(n) = T(n)^{1/m} \cdot \mathrm{polylog}(T(n))$.
- Decision complexity $d^{\mathsf{out}}(n) = T(n)^{1/m} \cdot \mathrm{polylog}(T(n))$.

Inner PCPP. Let $\delta^{\mathsf{in}} := \rho^{\mathsf{out}}/2$. By Theorem 7.3, there is a constant $s^{\mathsf{in}} \in (0,1)$ and a PCPP verifier V^{in} for Circuit-Eval$^{\perp}$ with randomness complexity $\log T(n) + O(\log\log T(n))$, soundness error s^{in}, proximity parameter δ^{in}, query complexity $q = 3$, and decision complexity $d^{\mathsf{in}} = \mathrm{poly}(\log\log T(n))$. Without loss of generality, we assume that the proof length is $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$.

Composition. We now compose V^{out} with the inner PCPP V^{in} by Theorem 6.10. We first check that the technical conditions are satisfied.

- $q^{\mathsf{in}} = 3 = O(1)$, $\rho^{\mathsf{out}} \geq \delta^{\mathsf{in}}$, $\ell^{\mathsf{in}} = 2^{r^{\mathsf{in}}}$.
- Since $r^{\mathsf{out}}_{\mathsf{col}} = w^{\mathsf{out}}_{\mathsf{proof}} - (4/m)\log T(n)$ and $r^{\mathsf{out}}_{\mathsf{shared}} \geq (7/m)\log T(n)$, we know that $r^{\mathsf{out}}_{\mathsf{col}} \leq w^{\mathsf{out}}_{\mathsf{proof}} \leq r^{\mathsf{out}}_{\mathsf{col}} + r^{\mathsf{out}}_{\mathsf{shared}}$.

By Theorem 6.10, we can obtain a rectangular PCPP V^{comp} with ROP, whose parameters are as follows:

- Soundness error $s^{\mathsf{comp}} = 1 - (1 - s^{\mathsf{in}}) \cdot (1 - s^{\mathsf{out}}) < 1$.
- Proximity parameter $\delta^{\mathsf{comp}} = \delta^{\mathsf{out}} = \delta$.
- Query complexity and ROP parity checking complexity $q^{\mathsf{comp}} = q = 3$.
- Proof matrix width $W_{\mathsf{proof}} = W^{\mathsf{out}}_{\mathsf{proof}} = 2^{w_{\mathsf{proof}}}$.

- Proof matrix height $H_{\mathsf{proof}} = H^{\mathsf{out}}_{\mathsf{proof}} + 2^{r^{\mathsf{out}}(n)+r^{\mathsf{in}}(d^{\mathsf{out}}(n))}/W^{\mathsf{out}}_{\mathsf{proof}}$. Note that

$$\begin{aligned} H^{\mathsf{out}}_{\mathsf{proof}} &= T(n)\log^{\Theta(m)} T(n)/W^{\mathsf{out}}_{\mathsf{proof}}(n), \\ r^{\mathsf{out}} &= (1-1/m)\log T(n) + \Theta(m \log\log T(n)), \\ r^{\mathsf{in}}(d^{\mathsf{out}}) &= (1/m)\log T(n) + O(\log\log(T(n))), \end{aligned}$$

  hence $H_{\mathsf{proof}} = T(n)\cdot\log^{\Theta(m)}(T(n))/W_{\mathsf{proof}}$. Without loss of generality, we assume that H_{proof} is a power of two. We then define

$$h_{\mathsf{proof}} := \log H_{\mathsf{proof}} = \log T(n) + \Theta(m\log\log T(n)) - w_{\mathsf{proof}}.$$

- Decision complexity $d^{\mathsf{in}}(d^{\mathsf{out}}(n)) = \mathrm{poly}(\log\log T(n))$.

Now we determine the randomness complexity of the composed PCPP verifier. Note that

$$\begin{aligned} r_{\mathsf{shared}} &= r^{\mathsf{out}}_{\mathsf{shared}} + r^{\mathsf{in}} \\ &= (7/m)\log T(n) + O(\log\log T(n) + m\log m) + (1/m)\log T(n) + O(\log\log T(n)) \\ &= (8/m)\log T(n) + O(\log\log T(n) + m\log m), \\ r_{\mathsf{row}} &= r^{\mathsf{out}}_{\mathsf{row}} = h_{\mathsf{proof}} - (4/m)\log T(n) - \Theta(m\log\log T(n)) \geq h_{\mathsf{proof}} - (5/m)\log T(n), \\ r_{\mathsf{col}} &= r^{\mathsf{out}}_{\mathsf{col}} = w_{\mathsf{proof}} - (4/m)\log T(n). \end{aligned}$$

Since we can always move some portion of $\mathsf{seed.row}$ or $\mathsf{seed.col}$ into $\mathsf{seed.shared}$, we can simply assume that $r_{\mathsf{row}} = h_{\mathsf{proof}} - (5/m)\log T(n)$, $r_{\mathsf{col}} = w_{\mathsf{proof}} - (5/m)\log T(n)$, and $r_{\mathsf{shared}} = (10/m)\log T(n) + O(\log\log T(n) + m\log m)$.

Moreover, by Remark 6.11 and the fact that the decision predicate of V^{in} is an OR of the answers or their negations (see Theorem 7.3), we know that the total number of queries and parity-check bits of V^{comp} is at most 3, and that for every $\mathsf{seed.shared}$, the decision predicate $\mathsf{VDec} \leftarrow V^{\mathsf{comp}}_{\mathsf{dec}}(\mathsf{seed.shared})$ of V^{comp} is an OR of its input bits (i.e., query answers and parity-check bits) or their negations. □

7.3 A 2-Query Rectangular PCPP with Imperfect Completeness

Following the construction in [CW19b, Appendix A], we can also construct a 2-query rectangular PCPP with a constant gap between the completeness and soundness parameters, using the following classical gadget due to [GJS76].

Lemma 7.5. *Let $x_1, x_2, x_3 \in \{0,1\}$ be Boolean variables. If $x_1 \vee x_2 \vee x_3$, then there is an $y \in \{0,1\}$ such that at least 7 of the following 10 constraints are satisfied:*

$$x_1, x_2, x_3, \overline{x_1} \vee \overline{x_2}, \overline{x_1} \vee \overline{x_3}, \overline{x_2} \vee \overline{x_3}, y, x_1 \vee \overline{y}, x_2 \vee \overline{y}, x_3 \vee \overline{y}. \tag{14}$$

Otherwise, at most 6 of the constraints in Eq. (14) are satisfied for any $y \in \{0,1\}$. Moreover, every $x_1, x_2, x_3, y \in \{0,1\}$ satisfies at most 7 of the above 10 constraints.

Theorem 7.6 (2-Query Rectangular PCPP)**.** *For every constant $\delta \in (0,1)$, there are constants $0 < s < c < 1$ such that the following holds. Let $m = m(n)$, $T(n)$, $w_{\mathsf{proof}}(n)$, $w_{\mathsf{input}}(n)$ be good functions such that $1 \leq m \leq (\log T(n))^{0.1}$, $n \leq T(n) \leq 2^{\mathrm{poly}(n)}$, $w_{\mathsf{proof}}(n) \leq \log T(n)$, and $w_{\mathsf{input}}(n) \leq \log n$. Then there are good functions $h_{\mathsf{proof}}(n)$ and $h_{\mathsf{input}}(n)$ satisfying*

$$\begin{aligned} &h_{\mathsf{proof}}(n) = \log T(n) + \Theta(m\log\log T(n)) - w_{\mathsf{proof}}(n), && \text{and} \\ &h_{\mathsf{input}}(n) = \lceil \log n \rceil - w_{\mathsf{input}}(n), \end{aligned}$$

such that the following holds.

Suppose that $w_{\mathsf{proof}}, h_{\mathsf{proof}} \geq (5/m)\log T(n)$*, and that for some absolute constant* $C \geq 1$*,*

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

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 9.*

Furthermore, given the randomness $\mathsf{seed} \in \{0,1\}^r$*, the total number of queries and parity-check bits is at most* 2*, and the decision predicate* $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ *of the rectangular PCPP verifier is an* OR *of the* 2 *input bits (including queries and parity-check bits) or their negations for every* seed.shared*.*

Completeness error	$1 - c$
Soundness error	s
Proximity parameter	δ
Row randomness	$h_{\mathsf{proof}} - (5/m)\log T(n)$
Column randomness	$w_{\mathsf{proof}} - (5/m)\log T(n)$
Shared randomness	$(10/m)\log T(n) + O(\log\log T(n) + m\log m)$
Query complexity	2
Parity check complexity	
Decision complexity	$\mathrm{poly}(\log\log T(n))$

Table 9: Parameters of the PCPP constructed in Theorem 7.6.

Proof. Let δ be an arbitrary constant. By Theorem 7.4, that there is a rectangular PCPP verifier V^{3q} with perfect completeness and parameters:

- Soundness error $s^{\mathsf{3q}} \in (0,1)$.
- Proximity parameter δ.
- Query complexity and parity-check complexity 3.
- Proof matrix size $H^{\mathsf{3q}}_{\mathsf{proof}} \times W^{\mathsf{3q}}_{\mathsf{proof}}$, with $w^{\mathsf{3q}}_{\mathsf{proof}} = w_{\mathsf{proof}} = \log W_{\mathsf{proof}}$ and $h^{\mathsf{3q}}_{\mathsf{proof}} = \log H^{\mathsf{3q}}_{\mathsf{proof}} = \log T(n) + \Theta(m\log\log T(n)) - w_{\mathsf{proof}}(n)$.[28]
- Shared randomness complexity $r_{\mathsf{shared}} = (10/m)\log T(n) + O(\log\log T(n) + m\log m)$.
- Row randomness complexity $r_{\mathsf{row}} = h^{\mathsf{3q}}_{\mathsf{proof}} - (5/m)\log T(n)$.
- Column randomness complexity $r_{\mathsf{col}} = w^{\mathsf{3q}}_{\mathsf{proof}} - (5/m)\log T(n)$.
- Decision complexity $\mathrm{poly}(\log\log T(n))$.

Let $r = r_{\mathsf{row}} + r_{\mathsf{col}} + r_{\mathsf{shared}}$ be the length of total randomness. Moreover, we know that the total number of queries and parity-check bits is at most 3, and that the decision circuit of V is an OR of its input bits (i.e. the answers to the queries and parity-check bits) or their negations after fixing the random seed. We will now combine V^{3q} and the gadget in Lemma 7.5 to construct a 2-query PCPP.

[28] Note that the final matrix height is $h_{\mathsf{proof}} \leq h^{\mathsf{3q}}_{\mathsf{proof}} + O(\log\log T(n))$, hence the technical requirement $h_{\mathsf{input}}(n)/h^{\mathsf{3q}}_{\mathsf{proof}} \leq 1 - C'\log\log T(n)/\log T(n)$ for large C' holds, given the assumption that $h_{\mathsf{input}}(n)/h_{\mathsf{proof}} \leq 1 - C\log\log T(n)/\log T(n)$ for large C, $h_{\mathsf{proof}} \geq (5/m)\log T(n)$, and $m \leq (\log T(n))^{0.1}$, as shown in Claim 6.17.

Suppose, for the simplicity of presentation, that the PCPP verifier V always probes 2 bits of the input and proof oracles, and has 1 parity-check bit. (The other cases can be considered similarly and we omit the details.) Then the decision predicate $\mathsf{VDec} \leftarrow V_{\mathsf{dec}}(\mathsf{seed.shared})$ for every fixed $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ is a function

$$\mathsf{VDec}(\mathsf{ans}_1, \mathsf{ans}_2, pc_1(\mathsf{seed})) := (\mathsf{ans}_1 \oplus b_1) \vee (\mathsf{ans}_2 \oplus b_2) \vee (pc_1(\mathsf{seed}) \oplus b_3).$$

where $b_1, b_2, b_3 \in \{0,1\}$. The new PCPP verifier V is defined as follows.

- The proof of the new PCPP verifier V is the concatenation of the proof for V^{3q} and an $y : \{0,1\}^r \to \{0,1\}$ of length 2^r used as the additional variable y in Lemma 7.5.
- The randomness of V is the concatenation of the randomness seed for V^{3q} and an $j \in [10]$.

Queries and parity-check bits. Assume that $(\mathsf{seed}, j) \in \{0,1\}^r \times [10]$ is given as the randomness. The verifier V first generates the indices i_1, i_2 of the queries to the input and proof oracles (denoted by a single oracle Π for simplicity) and the parity-check function pc_1. Instead of making all these queries and doing the parity-check, we identify $\mathsf{ans}_1 \oplus b_1, \mathsf{ans}_2 \oplus b_2, pc_1(\mathsf{seed}) \oplus b_3, y(\mathsf{seed})$ with x_1, x_2, x_3, y in the gadget given by Lemma 7.5, respectively, and queries the j-th gadget. (For instance, if $j = 5$, the corresponding constraint is $\overline{x_1} \vee \overline{x_3}$, so that we will query the i_1-th of Π and do the parity-check pc_1; if $j = 8$, the constraint is $x_1 \vee \overline{y}$, so that we will query the i_1-th bit of Π and the $y(\mathsf{seed})$.) The decision predicate will accept if and only if either the j-th constraint is satisfied when identifying $\mathsf{ans}_1 \oplus b_1, \mathsf{ans}_2 \oplus b_2, pc_1(\mathsf{seed}) \oplus b_3, y(\mathsf{seed})$ with x_1, x_2, x_3, y, respectively.

Completeness. For every input $x \in L$, by the completeness of V^{3q}, there is a proof oracle $\Pi^{\mathsf{3q}}_{\mathsf{proof}}$ such that V^{3q} accepts given the oracle $x \circ \Pi^{\mathsf{3q}}_{\mathsf{proof}}$ with probability 1, which means that for every $\mathsf{seed} \in \{0,1\}^r$, the answers $\mathsf{ans}_1, \mathsf{ans}_2$ to the queries and the parity-check bits $pc_1(\mathsf{seed})$ satisfies

$$\mathsf{VDec}(\mathsf{ans}_1, \mathsf{ans}_2, pc_1(\mathsf{seed})) = (\mathsf{ans}_1 \oplus b_1) \vee (\mathsf{ans}_2 \oplus b_2) \vee (pc_1(\mathsf{seed}) \oplus b_3) = 1.$$

By Lemma 7.5, there is an y_{seed} such that at least 7 of the 10 constraints in the gadgets are satisfied. This means that given the proof oracle $\Pi \circ y$ for $y(\mathsf{seed}) := y_{\mathsf{seed}}$, the verifier will accept with probability at least $c := 7/10$.

Soundness. Assume that $x \in \{0,1\}^n$ that is δ-far from being in L, and $\Pi_{\mathsf{proof}} = \Pi^{\mathsf{3q}}_{\mathsf{proof}} \circ y$ is any proof, where $\Pi^{\mathsf{3q}}_{\mathsf{proof}}$ is a proof for V^{3q} and $y : \{0,1\}^r \to \{0,1\}$. By the soundness of V^{3q}, we know that for each least $1 - s_1$ fraction of $\mathsf{seed} \in \{0,1\}^n$,

$$\mathsf{VDec}(\mathsf{ans}_1, \mathsf{ans}_2, pc_1(\mathsf{seed})) = (\mathsf{ans}_1 \oplus b_1) \vee (\mathsf{ans}_2 \oplus b_2) \vee (pc_1(\mathsf{seed}) \oplus b_3) = 0.$$

By Lemma 7.5, we can see that for these seed, the accept probability of V is at most $6/10$, whereas in other cases the accept probabilistic of V is at most $7/10$. Thus the accept probability of V is at most $s := (7/10) \cdot s_1 + (6/10) \cdot (1 - s_1) < c$.

Rectangularity. Since we only need to introduce $O(1)$ bits of randomness representing $j \sim [10]$, we can put it into the shared randomness. We only need to show that the new proof $\Pi^{\mathsf{3q}}_{\mathsf{proof}} \circ y$ can be arranged as a matrix so that the queries can be done rectangularly. Let $W_{\mathsf{proof}} := W^{\mathsf{3q}}_{\mathsf{proof}}$ and

$H_{\mathsf{proof}} := H^{\mathsf{3q}}_{\mathsf{proof}} + 2^r/W^{\mathsf{3q}}_{\mathsf{proof}}$. Without loss of generality, we assume that H_{proof} is a power of two, therefore we define

$$\begin{aligned} h_{\mathsf{proof}} := \log H_{\mathsf{proof}} &= \log\left(H^{\mathsf{3q}}_{\mathsf{proof}} + 2^r/W^{\mathsf{3q}}_{\mathsf{proof}}\right) \\ &\leq h^{\mathsf{3q}}_{\mathsf{proof}} + O(\log\log T(n)) \\ &= \log T(n) + \Theta(m\log\log T(n)) - w_{\mathsf{proof}}. \end{aligned}$$

The final proof matrix will be of size $H_{\mathsf{proof}} \times W_{\mathsf{proof}}$, arranged as follows: The first $H^{\mathsf{3q}}_{\mathsf{proof}}$ rows will contain the proof $\Pi^{\mathsf{3q}}_{\mathsf{proof}}$ of V^{3q}; The remaining $2^r/W_{\mathsf{proof}}$ contains the proof $y : \{0,1\}^r \to \{0,1\}$, represented as the string $y(0) \circ y(1) \circ \cdots \circ y(2^r-1)$ of length 2^r. Recall that there will be two kinds of queries to the proof oracle.

1. If the query is to the proof oracle $\Pi^{\mathsf{3q}}_{\mathsf{proof}}$ of V^{3q} or to the input oracle, we can use the row and column verifier of V^{3q} to generate the queries rectangularly.
2. Otherwise, the query is to the proof $y(\mathsf{seed})$ for the randomness $\mathsf{seed} \in \{0,1\}^r$ of V^{3q}. Then the column (resp. row) index of this query only depends on the lowest w_{proof} bits (resp. the highest $r - w_{\mathsf{proof}}$ bits) of the random seed of V. Recall that the random seed of V is the concatenation of seed and a $j \in [10]$. If we arrange the randomness as

   $$\mathsf{seed.col} \circ \mathsf{seed.shared} \circ j \circ \mathsf{seed.row},$$

   then the lowest w_{proof} bits (resp. the highest $r - w_{\mathsf{proof}}$ bits) of the random seed only depends on the $(\mathsf{seed.col}, \mathsf{seed.shared})$ (resp. $(\mathsf{seed.shared}, j, \mathsf{seed.row})$), since

   $$\begin{aligned} r_{\mathsf{col}} &= w_{\mathsf{proof}} - (5/m)\log T(n) \leq w_{\mathsf{proof}} \\ r_{\mathsf{col}} + r_{\mathsf{shared}} &= w_{\mathsf{proof}} + (5/m)\log T(n) + O(\log\log n + m\log m) \geq w_{\mathsf{proof}}. \end{aligned}$$

As a result, the queries can be done rectangularly. □

Acknowledgements

We got the initial idea of this work when Jiatu was a research intern at Igor Carboni Oliveira's group and Hanlin was visiting Igor. Hanlin wants to thank his supervisor Rahul Santhanam for introducing the problems Partial-Hard [AS10] and XOR-Remote-Point [APY09] to him and for helpful discussions. We thank Lijie Chen and an anonymous STOC reviewer for pointing out that the technique in [Wil18c] can be slightly adapted to obtain a non-trivial algorithm for #ACC-Satisfying-Pairs. We also thank Zhikun Wang and Tianqi Yang for discussion on this work.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264. (cit. on p. 1, 74)

[AC19] Josh Alman and Lijie Chen. Efficient construction of rigid matrices using an NP oracle. In *FOCS*, pages 1034–1055. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00067. (cit. on p. 1, 7, 9, 53, 54)

[ACW16] Josh Alman, Timothy M. Chan, and R. Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In *FOCS*, pages 467–476. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.57. (cit. on p. 11)

[ACW20] Josh Alman, Timothy M. Chan, and R. Ryan Williams. Faster deterministic and Las Vegas algorithms for offline approximate nearest neighbors in high dimensions. In *SODA*, pages 637–649. SIAM, 2020. doi:10.1137/1.9781611975994.39. (cit. on p. 11)

[AG91] Eric Allender and Vivek Gore. On strong separations from AC^0 (extended abstract). In *Fundamentals of Computation Theory, 8th International Symposium, FCT '91*, volume 529 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1991. doi:10.1007/3-540-54458-5_44. (cit. on p. 7, 53)

[AHWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *STOC*, pages 375–388. ACM, 2016. doi:10.1145/2897518.2897653. (cit. on p. 5)

[Ajt83] Miklós Ajtai. Σ^1_1-formulae on finite structures. *Ann. Pure Appl. Log.*, 24(1):1–48, 1983. doi:10.1016/0168-0072(83)90038-6. (cit. on p. 4)

[APY09] Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the nearest codeword problem. In *APPROX-RANDOM*, volume 5687 of *Lecture Notes in Computer Science*, pages 339–351. Springer, 2009. doi:10.1007/978-3-642-03685-9_26. (cit. on p. 3, 86)

[AS10] Vikraman Arvind and Srikanth Srinivasan. Circuit lower bounds, help functions, and the remote point problem. In *ICS*, pages 383–396. Tsinghua University Press, 2010. URL: http://conference.iiis.tsinghua.edu.cn/ICS2010/content/papers/30.html. (cit. on p. 3, 4, 8, 56, 86)

[AW15] Josh Alman and R. Ryan Williams. Probabilistic polynomials and Hamming nearest neighbors. In *FOCS*, pages 136–150. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.18. (cit. on p. 11)

[AW17] Josh Alman and R. Ryan Williams. Probabilistic rank and matrix rigidity. In *STOC*, pages 641–652. ACM, 2017. doi:10.1145/3055399.3055484. (cit. on p. 11)

[AWY15] Amir Abboud, R. Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *SODA*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17. (cit. on p. 11)

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. 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. 9, 15, 16, 57, 68, 69, 73, 80)

[BHPT20] Amey Bhangale, Prahladh Harsha, Orr Paradise, and Avishay Tal. Rigid matrices from rectangular PCPs or: Hard claims have complex proofs. In *FOCS*, pages 858–869. IEEE, 2020. doi:10.1109/FOCS46700.2020.00084. (cit. on p. 1, 9, 17, 54, 57, 58, 59, 61, 63, 64, 65, 69)

[BT94] Richard Beigel and Jun Tarui. On ACC. *Comput. Complex.*, 4:350–366, 1994. doi:10.1007/BF01263423. (cit. on p. 7, 53)

[Che19] Lijie Chen. Non-deterministic quasi-polynomial time is average-case hard for ACC circuits. In *FOCS*, pages 1281–1304. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00079. (cit. on p. 2, 3)

[Che22] Lijie Chen. *Better Hardness via Algorithms, and New Forms of Hardness versus Randomness*. PhD thesis, Massachusetts Institute of Technology, 2022. (cit. on p. 5)

[CJW20] Lijie Chen, Ce Jin, and R. Ryan Williams. Sharp threshold results for computational complexity. In *STOC*, pages 1335–1348. ACM, 2020. doi:10.1145/3357713.3384283. (cit. on p. 12)

[CL21] Lijie Chen and Xin Lyu. Inverse-exponential correlation bounds and extremely rigid matrices from a new derandomized XOR lemma. In *STOC*, pages 761–771. ACM, 2021. doi:10.1145/3406325.3451132. (cit. on p. 3, 6, 54)

[CLW20] Lijie Chen, Xin Lyu, and R. Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *FOCS*, pages 1–12. IEEE, 2020. doi:10.1109/FOCS46700.2020.00009. (cit. on p. ii, 2, 3, 6, 8, 10, 13, 14, 15, 35, 54, 55, 100)

[Cop82] Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982. doi:10.1137/0211037. (cit. on p. 11, 52)

[COS18] Ruiwen Chen, Igor Carboni Oliveira, and Rahul Santhanam. An average-case lower bound against ACC^0. In *LATIN*, volume 10807 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2018. doi:10.1007/978-3-319-77404-6_24. (cit. on p. 3)

[CR22] Lijie Chen and Hanlin Ren. Strong average-case circuit lower bounds from nontrivial derandomization. *SIAM J. Comput.*, 51(3):STOC20–115–STOC20–173, 2022. doi:10.1137/20M1364886. (cit. on p. 3, 35)

[CW19a] Lijie Chen and Ruosong Wang. Classical algorithms from quantum and Arthur-Merlin communication protocols. In *ITCS*, volume 124 of *LIPIcs*, pages 23:1–23:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.23. (cit. on p. 5, 11)

[CW19b] Lijie Chen and R. Ryan Williams. Stronger connections between circuit analysis and circuit lower bounds, via PCPs of proximity. In *CCC*, volume 137 of *LIPIcs*, pages 19:1–19:43. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CCC.2019.19. (cit. on p. 2, 7, 9, 22, 32, 35, 80, 83, 98)

[CW21] Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926. (cit. on p. 7, 11, 53)

[CZ19] Eshan Chattopadhyay and David Zuckerman. Explicit two-source extractors and resilient functions. *Annals of Mathematics*, 189(3):653–705, 2019. doi:10.4007/annals.2019.189.3.1. (cit. on p. 1)

[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 *FOCS*, pages 89–98. IEEE Computer Society, 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 *CCC*, volume 50 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CCC.2016.19. (cit. on p. 13, 14)

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431. (cit. on p. 4)

[GGH+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *STOC*, pages 440–449. ACM, 2007. doi:10.1145/1250790.1250855. (cit. on p. 9)

[GGNS23] Karthik Gajulapalli, Alexander Golovnev, Satyajeet Nagargoje, and Sidhant Saraogi. Range avoidance for constant-depth circuits: Hardness and algorithms. *CoRR*, 2023. doi:10.48550/arXiv.2303.05044. (cit. on p. 2)

[GJS76] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1. (cit. on p. 83)

[GLW22] Venkatesan Guruswami, Xin Lyu, and Xiuhan Wang. Range avoidance for low-depth circuits and connections to pseudorandomness. In *APPROX/RANDOM*, volume 245 of *LIPIcs*, pages 20:1–20:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.20. (cit. on p. 2, 10, 11)

[GNW11] Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR-lemma. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, volume 6650 of *Lecture Notes in Computer Science*, pages 273–301. Springer, 2011. doi:10.1007/978-3-642-22670-0_23. (cit. on p. 15)

[Gol11] Oded Goldreich. *A Sample of Samplers: A Computational Perspective on Sampling*, pages 302–332. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-22670-0_24. (cit. on p. 60)

[GR08] Dan Gutfreund and Guy N. Rothblum. The complexity of local list decoding. In *APPROX-RANDOM*, 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. 10)

[Hås89] Johan Håstad. Almost optimal lower bounds for small depth circuits. *Adv. Comput. Res.*, 5:143–170, 1989. (cit. on p. 4)

[HV21] Xuangui Huang and Emanuele Viola. Average-case rigidity lower bounds. In *CSR*, volume 12730 of *Lecture Notes in Computer Science*, pages 186–205. Springer, 2021. doi:10.1007/978-3-030-79416-3_11. (cit. on p. 54)

[KKMP21] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos H. Papadimitriou. Total functions in the polynomial hierarchy. In *ITCS*, volume 185 of *LIPIcs*, pages 44:1–44:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITCS.2021.44. (cit. on p. 1, 3)

[Kor21] Oliver Korten. The hardest explicit construction. In *FOCS*, pages 433–444. IEEE, 2021. doi:10.1109/FOCS52979.2021.00051. (cit. on p. 1, 10)

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Comb.*, 7(4):357–363, 1987. doi:10.1007/BF02579323. (cit. on p. 15)

[Li23] Xin Li. Two source extractors for asymptotically optimal entropy, and (many) more. *CoRR*, abs/2303.06802, 2023. arXiv:2303.06802, doi:10.48550/arXiv.2303.06802. (cit. on p. 1)

[LY22] Jiatu Li and Tianqi Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. In *STOC*, pages 1180–1193. ACM, 2022. doi:10.1145/3519935.3519976. (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. 76)

[Mul11] Ketan Mulmuley. On P vs. NP and geometric complexity theory: Dedicated to Sri Ramakrishna. *J. ACM*, 58(2):5:1–5:26, 2011. doi:10.1145/1944345.1944346. (cit. on p. 12)

[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. 2)

[Par21] Orr Paradise. Smooth and strong PCPs. *Comput. Complex.*, 30(1):1, 2021. doi:10.1007/s00037-020-00199-3. (cit. on p. 10, 18)

[Păt08] Mihai Pătraşcu. Succincter. In *FOCS*, pages 305–313. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.83. (cit. on p. 11)

[Pip79] Nicholas Pippenger. On simultaneous resource bounds (preliminary version). In *FOCS*, pages 307–311. IEEE Computer Society, 1979. doi:10.1109/SFCS.1979.29. (cit. on p. 4)

[Ram20] C. Ramya. Recent progress on matrix rigidity - A survey. *CoRR*, abs/2009.09460, 2020. doi:10.48550/arXiv.2009.09460. (cit. on p. 1)

[Raz87] Alexander A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987. doi:10.1007/BF01137685. (cit. on p. 11)

[RSW22] Hanlin Ren, Rahul Santhanam, and Zhikun Wang. On the range avoidance problem for circuits. In *FOCS*, pages 640–650. IEEE, 2022. doi:10.1109/FOCS54457.2022.00067. (cit. on p. 2, 3, 4, 5, 9, 10, 11, 14, 16, 17, 57, 58, 63, 64, 65, 66, 68, 69, 70, 73, 76, 98)

[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)

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *STOC*, pages 77–82. ACM, 1987. doi:10.1145/28395.28404. (cit. on p. 11)

[Smo93] Roman Smolensky. On representations by low-degree polynomials. In *FOCS*, pages 130–138. IEEE Computer Society, 1993. doi:10.1109/SFCS.1993.366874. (cit. on p. 11)

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Inf. Theory*, 42(6):1723–1731, 1996. doi:10.1109/18.556668. (cit. on p. 13)

[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. 74)

[Val77] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *MFCS*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1977. doi:10.1007/3-540-08353-7_135. (cit. on p. 1, 9)

[VW18] Emanuele Viola and Avi Wigderson. Local expanders. *Comput. Complex.*, 27(2):225–244, 2018. doi:10.1007/s00037-017-0155-1. (cit. on p. 73)

[Wil13] R. Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013. doi:10.1137/10080703X. (cit. on p. 2)

[Wil14a] R. Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014. doi:10.1145/2559903. (cit. on p. 2, 52)

[Wil14b] R. Ryan Williams. The polynomial method in circuit complexity applied to algorithm design (invited talk). In *FSTTCS*, volume 29 of *LIPIcs*, pages 47–60. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.47. (cit. on p. 11)

[Wil18a] R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524. (cit. on p. 11, 52)

[Wil18b] R. Ryan Williams. Limits on representing Boolean functions by linear combinations of simple functions: Thresholds, ReLUs, and low-degree polynomials. In *CCC*, volume 102 of *LIPIcs*, pages 6:1–6:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CCC.2018.6. (cit. on p. 32, 35)

[Wil18c] 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. 7, 12, 53, 86)

[Yao85] Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *FOCS*, pages 1–10. IEEE Computer Society, 1985. doi:10.1109/SFCS.1985.49. (cit. on p. 4)

A Missing Proofs in Section 3 and 4

In this section, we complete the missing proofs in the framework of solving range avoidance, remote point, and hard partial truth table via non-trivial satisfying pairs algorithms.

A.1 Satisfying Pairs for $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ Circuits

Theorem 3.7. *Let $\mathscr{C}$ be a typical circuit class, $M' \geq 1$ and $\eta \in (0,1)$ be parameters. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} = T^{\mathrm{alg}}(N, M)$ that, given as input a list of $\hat{N} \leq N$ $\mathsf{AND}_d \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of $\hat{M} \leq M$ inputs $\{x_j\}$, estimates the following quantity with additive error η:*

$$\Pr_{i \leftarrow [\hat{N}], j \leftarrow [\hat{M}]} [C_i(x_j)].$$

Then, there is a deterministic algorithm running in time $A^d(2^{dl} + M'/M) \cdot (\ell_{\mathsf{Prod}}/N) \cdot O(T^{\mathrm{alg}})$ that, given as input a $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit C^{Prod} with parameters specified in Remark 3.6, and a list of M' inputs $\{(x_j, \alpha_j)\}$, estimates the following quantity with additive error $\eta \cdot U^d$:

$$\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow [M']} \left[C_i^{\mathsf{Prod}}(x_j, \alpha_j) \right].$$

Proof. For any fixed i and j, we know that

$$\begin{aligned} C_i^{\mathsf{Prod}}(x_j, \alpha_j) &= \prod_{t=1}^{d} \mathsf{Sum}_{q_t(i)}(x_j, \alpha_j) \\ &= \prod_{t=1}^{d} \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_j) \cdot C_{\mathsf{idx}_k(\alpha_j, q_t(i))}(x_j) \\ &= \sum_{k_1 \in [A]} \sum_{k_2 \in [A]} \cdots \sum_{k_d \in [A]} \prod_{t=1}^{d} \Big(\mathsf{coeff}_{k_t}(\alpha_j) \cdot C_{\mathsf{idx}_{k_t}(\alpha_j, q_t(i))}(x_j) \Big). \end{aligned} \tag{15}$$

As we can enumerate $k_1, k_2, \ldots, k_d \in [A]$ in A^d time, it suffices to estimate

$$\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow [M']} \left[\prod_{t=1}^{d} \Big(\mathsf{coeff}_{k_t}(\alpha_j) \cdot C_{\mathsf{idx}_{k_t}(\alpha_j, q_t(i))}(x_j) \Big) \right]. \tag{16}$$

Fix $k_1, k_2, \ldots, k_d \in [A]$. Since C^{Prod} is of locality l, we can see that Eq. (16) only depends on dl bits of α_j. We partition $j \in [M']$ into 2^{dl} groups as follows: For each $\alpha \in \{0,1\}^{dl}$, let $\mathcal{J}_\alpha$ be the set of $j \in [M']$ such that the dl bits of α_j (that Eq. (16) for this j depends on) equals to α. We will estimate Eq. (16) by enumerating $\alpha \in \{0,1\}^{dl}$, estimating it for $j \leftarrow \mathcal{J}_\alpha$ (instead of $j \leftarrow [M']$), and then taking the average over all possible α.

Now we fix any $\alpha \in \{0,1\}^{dl}$. We can rephrase the following two items as they no longer depend on α_j:

$$\begin{aligned} \mathsf{coeff}_{k_t}(\alpha_j) &=: \mathsf{coeff}'_t; \\ C_{\mathsf{idx}_{k_t}(\alpha_j, q_t(i))}(x_j) &=: C_{\mathsf{idx}'_t(i)}(x_j). \end{aligned}$$

It then suffices to estimate

$$\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow \mathcal{J}_\alpha} \left[\prod_{t=1}^{d} \mathsf{coeff}'_t \cdot C_{\mathsf{idx}'_t(i)}(x_j) \right] = \left(\prod_{t=1}^{d} \mathsf{coeff}'_t \right) \cdot \mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow \mathcal{J}_\alpha} \left[\bigwedge_{t=1}^{d} C_{\mathsf{idx}'_t(i)}(x_j) \right]. \tag{17}$$

Each expression of the form $\mathbb{E}_{i,j} \left[\bigwedge_{t=1}^{d} C_{\mathsf{idx}'_t(i)}(x_j) \right]$ can be reduced to the Satisfying-Pairs problem for $\mathsf{AND}_d \circ \mathscr{C}$ circuits. More precisely, we split $\mathcal{J}_\alpha$ into blocks of size M, split $[\ell_{\mathsf{Prod}}]$ into

blocks of size N, and use the assumed algorithm (which works for N $\mathsf{AND}_d \circ \mathscr{C}$ circuits and M inputs) to estimate Eq. (17) within additive error $\left(\eta \cdot \prod_{t=1}^{d} |\mathsf{coeff}'_t|\right)$ in $T^{\mathrm{alg}} \cdot \lceil |\mathcal{J}_\alpha|/M \rceil \cdot \lceil \ell_{\mathsf{Prod}}/N \rceil$ time.[29] We substitute this estimation in Eq. (16) and then in Eq. (15) to obtain our final algorithm.

It remains to upper bound the running time and the additive error of our algorithm.

Running time. Consider the subroutine for estimating Eq. (16). This subroutine itself is reduced to subroutines for each $\mathcal{J}_\alpha$, which takes $T^{\mathrm{alg}} \cdot \lceil |\mathcal{J}_\alpha|/M \rceil \cdot \lceil \ell_{\mathsf{Prod}}/N \rceil$ time. The time complexity of this subroutine is

$$O(T^{\mathrm{alg}}) \cdot \lceil \ell_{\mathsf{Prod}}/N \rceil \cdot \sum_{\alpha} \lceil |\mathcal{J}_\alpha|/M \rceil \leq O(T^{\mathrm{alg}}) \cdot (2^{dl} + M'/M) \cdot (\ell_{\mathsf{Prod}}/N).$$

We invoked this subroutine A^d times by enumerating $k_1, k_2, \ldots, k_d \in [A]$ to estimate Eq. (15), so the total time complexity of our algorithm is $A^d(2^{dl} + M'/M)(\ell_{\mathsf{Prod}}/N)O(T^{\mathrm{alg}})$.

Additive error. Our estimation of Eq. (17) is within additive error $\left(\eta \cdot \prod_{t=1}^{d} |\mathsf{coeff}'_t|\right)$. Thus our estimation of Eq. (16) is within additive error of

$$\eta \cdot \sum_{\alpha \in \{0,1\}^{dl}} \frac{|\mathcal{J}_\alpha|}{M'} \cdot \prod_{i=1}^{d} |\mathsf{coeff}'_t| = \eta \cdot \mathop{\mathbb{E}}_{j} \left[\prod_{i=1}^{d} |\mathsf{coeff}_{k_t}(\alpha_j)| \right].$$

It follows that our estimation of Eq. (15) is within additive error of

$$\begin{aligned} &\eta \cdot \sum_{k_1 \in [A]} \sum_{k_2 \in [A]} \cdots \sum_{k_d \in [A]} \mathop{\mathbb{E}}_{j} \left[\prod_{i=1}^{d} |\mathsf{coeff}_{k_t}(\alpha_j)| \right] \\ =& \eta \cdot \mathop{\mathbb{E}}_{j} \left[\left(\sum_{k \in [A]} |\mathsf{coeff}_k(\alpha_j)| \right)^d \right] \\ \leq& \eta \cdot U^d. \end{aligned}$$

□

Lemma 4.5. *Let $\mathscr{C}$ be a typical circuit class, $M' \geq 1$ and $\eta \in (0,1)$ be parameters. Suppose there is a deterministic algorithm running in time $T^{\mathrm{alg}} = T^{\mathrm{alg}}(N, M)$ that, given as input a list of $\hat{M} \leq M$ $\mathsf{AND}_d \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of $\hat{N} \leq N$ inputs $\{x_j\}$ of length $n \cdot \mathrm{polylog}(\ell)$, estimates the following quantity with additive error η:*

$$\Pr_{i \leftarrow [\hat{M}], j \leftarrow [\hat{N}]} [C_i(x_j)].$$

Then, for any constant $\ell_{\mathscr{C}} > 0$, there is a deterministic algorithm running in time $A^d \cdot (\ell_{\mathscr{C}}^d + \ell_{\mathsf{Prod}}/N) \cdot (2^{dl} + M'/M) \cdot O(T^{\mathrm{alg}})$ that, given as input a $\mathsf{Prod}_d \circ \mathsf{Sum}$ circuit C^{Prod} with parameters specified in Remark 4.4, a list of ℓ_x strings $\{x_j\}$, a list of M' inputs $\{\alpha_j\}$, and a list of M' $\mathscr{C}$ circuits $\{C_j\}$ from $\{0,1\}^{|x|}$ to $\{0,1\}^{\ell_{\mathscr{C}}}$, estimates the following quantity with additive error $\eta \cdot U^d$:

$$\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow [M']} \left[C_i^{\mathsf{Prod}}(C_j(x), \alpha_j) \right].$$

Recall here that $C_j(x) = C_j(x_1) \circ C_j(x_2) \circ \cdots \circ C_j(x_{\ell_x})$.

[29] Note that at most one of the block may contain less than M inputs. However, the assumed algorithm works for input number $\leq M$ as well, and this will not have any blow-up on the error factor.

Proof Sketch of Lemma 4.5. We identify $\mathsf{idx}_k(\alpha, i) \in [\ell_y]$ with $(\mathsf{idx}^x_k(\alpha, i), \mathsf{idx}^{\mathscr{C}}_k(\alpha, i)) \in [\ell_x] \times [\ell_{\mathscr{C}}]$ (note that $\ell_y = \ell_x \ell_{\mathscr{C}}$). Then,

$$\begin{aligned} C^{\mathsf{Prod}}_i(C_j(x), \alpha_j) &= \prod_{t=1}^{d} \mathsf{Sum}_{q_t(i)}(C_j(x), \alpha_j) \\ &= \prod_{t=1}^{d} \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_j) \cdot (C_j)_{\mathsf{idx}^{\mathscr{C}}_k(\alpha_j, q_t(i))}(x_{\mathsf{idx}^x_k(\alpha_j, q_t(i))}) \\ &= \sum_{k_1 \in [A]} \sum_{k_2 \in [A]} \cdots \sum_{k_d \in [A]} \prod_{t=1}^{d} \Big(\mathsf{coeff}_{k_t}(\alpha_j) \cdot (C_j)_{\mathsf{idx}^{\mathscr{C}}_{k_t}(\alpha_j, q_t(i))}(x_{\mathsf{idx}^x_{k_t}(\alpha_j, q_t(i))}) \Big). \end{aligned} \tag{18}$$

As we can enumerate $k_1, k_2, \ldots, k_d \in [A]$ in A^d time, it suffices to estimate

$$\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow [M']} \left[\prod_{t=1}^{d} \Big(\mathsf{coeff}_{k_t}(\alpha_j) \cdot (C_j)_{\mathsf{idx}^{\mathscr{C}}_{k_t}(\alpha_j, q_t(i))}(x_{\mathsf{idx}^x_{k_t}(\alpha_j, q_t(i))}) \Big) \right]. \tag{19}$$

Fix $k_1, k_2, \ldots, k_d \in [A]$. Since C^{Prod} is of locality l, we can see that Eq. (19) only depends on dl bits of α_j. We partition $j \in [M']$ into 2^{dl} groups as follows: For each $\alpha \in \{0,1\}^{dl}$, let $\mathcal{J}_\alpha$ be the set of $j \in [M']$ such that the dl bits of α_j (that Eq. (19) for this j depends on) equals to α. We will estimate Eq. (19) by enumerating $\alpha \in \{0,1\}^{dl}$, estimating it for $j \leftarrow \mathcal{J}_\alpha$ (instead of $j \leftarrow [M']$), and then taking the average over all possible α.

Now we fix any $\alpha \in \{0,1\}^{dl}$. We can rephrase the following items as they no longer depend on α_j:

$$\begin{aligned} \mathsf{coeff}_{k_t}(\alpha_j) &=: \mathsf{coeff}'_t; \\ \mathsf{idx}^x_{k_t}(\alpha_j, q_t(i)) &=: \mathsf{idx}^{x'}_t(i); \\ \mathsf{idx}^{\mathscr{C}}_{k_t}(\alpha_j, q_t(i)) &=: \mathsf{idx}^{\mathscr{C}'}_t(i). \end{aligned}$$

It then suffices to estimate

$$\begin{aligned} &\mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow \mathcal{J}_\alpha} \left[\prod_{t=1}^{d} \mathsf{coeff}'_t \cdot (C_j)_{\mathsf{idx}^{\mathscr{C}'}_t(i)} \Big(x_{\mathsf{idx}^{x'}_t(i)} \Big) \right] \\ &= \left(\prod_{t=1}^{d} \mathsf{coeff}'_t \right) \cdot \mathop{\mathbb{E}}_{i \leftarrow [\ell_{\mathsf{Prod}}], j \leftarrow \mathcal{J}_\alpha} \left[\bigwedge_{t=1}^{d} (C_j)_{\mathsf{idx}^{\mathscr{C}'}_t(i)} \Big(x_{\mathsf{idx}^{x'}_t(i)} \Big) \right]. \end{aligned} \tag{20}$$

Now for $\beta \in [\ell_{\mathscr{C}}]^d$, let $\mathcal{I}_\beta := \{i \in [\ell_{\mathsf{Prod}}] : \forall t \in [d], \mathsf{idx}^{\mathscr{C}'}_t(i) = \beta_t\}$. We enumerate over β, and now it suffices to estimate

$$\mathop{\mathbb{E}}_{i \leftarrow \mathcal{I}_\beta, j \leftarrow \mathcal{J}_\alpha} \left[\bigwedge_{t=1}^{d} (C_j)_{\beta_t} \Big(x_{\mathsf{idx}^{x'}_t(i)} \Big) \right]. \tag{21}$$

Each expression of the form $\mathbb{E}_{i,j} \left[\bigwedge_{t=1}^{d} (C_j)_{\beta_t} \Big(x_{\mathsf{idx}^{x'}_t(i)} \Big) \right]$ can be reduced to the Satisfying-Pairs problem for $\mathsf{AND}_d \circ \mathscr{C}$ circuits. More precisely, we split $\mathcal{I}_\beta$ into blocks of size N and $\mathcal{J}_\alpha$ into blocks of size M, and use the assumed algorithm to estimate Eq. (21). By similar argument as in Theorem 3.7, the additive error of our algorithm is bounded by $\eta \cdot U^d$.

Complexity. The subroutine for estimating Eq. (21) takes $O(T^{\mathrm{alg}}) \cdot \lceil |\mathcal{J}_\alpha|/M \rceil \cdot \lceil |\mathcal{I}_\beta/N| \rceil$ time. Therefore, the subroutine for estimating Eq. (20) takes

$$\sum_{\beta \in [\ell_{\mathscr{C}}]^d} O(T^{\mathrm{alg}}) \cdot \lceil |\mathcal{J}_\alpha|/M \rceil \cdot \lceil |\mathcal{I}_\beta|/N \rceil = O(T^{\mathrm{alg}}) \cdot \lceil |\mathcal{J}_\alpha|/M \rceil \cdot (\ell_{\mathscr{C}}^d + \ell_{\mathsf{Prod}}/N)$$

time. It then follows that the subroutine for estimating Eq. (19) takes

$$\sum_{\alpha \in \{0,1\}^{dl}} O(T^{\mathrm{alg}}) \cdot \lceil |\mathcal{J}_\alpha|/M \rceil \cdot (\ell_{\mathscr{C}}^d + \ell_{\mathsf{Prod}}/N) = O(T^{\mathrm{alg}}) \cdot (2^{dl} + M'/M) \cdot (\ell_{\mathscr{C}}^d + \ell_{\mathsf{Prod}}/N)$$

time, and finally, estimating Eq. (18) takes

$$O(T^{\mathrm{alg}}) \cdot (2^{dl} + M'/M) \cdot (\ell_{\mathscr{C}}^d + \ell_{\mathsf{Prod}}/N) \cdot A^d$$

time, which is the total time complexity of our algorithm. ◇

A.2 Verifying PCPP with Satisfying Pairs

Lemma 3.8. *Let $\mathscr{C}$ be a typical circuit class and $d \geq 2$ be an even number. Suppose there is an algorithm that takes as input a list of $N = 2^{r_{\mathsf{col}}}$ $\mathsf{AND}_{2d} \circ \mathscr{C}$ circuits $\{C_i\}$ and a list of N inputs $\{x_j\}$, runs in deterministic T^{alg} time, and estimates the following quantity with additive error η:*

$$\Pr_{i,j \leftarrow [2^{r_{\mathsf{col}}}]}[C_i(x_j)].$$

Then there is an algorithm that takes the circuit C, $(w_1, w_2, \ldots, w_{\hat{H}_{\mathsf{proof}}})$, and $(\alpha_1, \alpha_2, \ldots, \alpha_{\hat{H}_{\mathsf{proof}}})$ as input, runs in deterministic $O((3A)^{2d} T^{\mathrm{alg}}) \cdot \left(2^{2dl + r_{\mathsf{shared}}} + T \log^{O(m)} T / 2^{2r_{\mathsf{col}}}\right)$ time, and satisfies the following:

(Completeness) *If for every $i \in [\hat{H}_{\mathsf{proof}}]$, it holds that (1) for every $j \in [W_{\mathsf{proof}}]$, $\pi_{i,j}^{\mathsf{Real}} \in [0,1]$; (2) $\|\pi_i^{\mathsf{Real}} - \pi_i^{\mathsf{Bool}}\|_1 \leq \delta$, then the algorithm accepts.*

(Soundness) *If the algorithm accepts, then it holds that*

1. *for every $\mathsf{seed.shared} \in \{0,1\}^{r_{\mathsf{shared}}}$ and $\iota \in [q]$, $\|f_{\mathsf{seed.shared},\iota}^{\mathsf{Real}}\|_d^d \leq 1 + 2\eta \cdot U^d$;*
2. *$\mathbb{E}_{i \leftarrow [\hat{H}_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]} \left[|\pi_{i,j}^{\mathsf{Real}} - \pi_{i,j}^{\mathsf{Bool}}|^d \right] \leq 4^d \cdot \delta + 2^{d+1} \eta (2U+1)^{2d}$.*

Proof. Fix $\mathsf{seed.shared}$ and ι, we first estimate

$$\|f_{\mathsf{seed.shared},\iota}^{\mathsf{Real}}\|_d^d := \mathop{\mathbb{E}}_{\mathsf{seed.row},\mathsf{seed.col}} \left[|\pi_{\mathsf{irow}[\iota],\mathsf{icol}[\iota]}^{\mathsf{Real}}|^d \right].$$

Recall that

$$\pi_{i,j}^{\mathsf{Real}} = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha_i) \cdot C_{\mathsf{idx}_k(\alpha_i, j)}(w_i).$$

Therefore, we can build a $\mathsf{Prod}_d \circ \mathsf{Sum} \circ \mathscr{C}$ circuit $C_{\mathsf{norm}} := C_{\mathsf{norm}}(\mathsf{seed.shared}, \iota)$ as follows.

Circuit C_{norm}

(Inputs) The input consists of (w, α) with the intended meaning that $w = w_{\mathsf{irow}[\iota]}$ and $\alpha = \alpha_{\mathsf{irow}[\iota]}$.

(Bottom circuits) The bottom circuit is exactly C (taking input w). Thus, there are ℓ output gates of $\mathscr{C}$ circuits with the i-th one being precisely the i-th output gate of C.

(Intermediate linear sum gates) There are $2^{r_{\mathsf{col}}}$ intermediate linear sum gates. For each $\mathsf{seed.col}$,

$$\mathsf{Sum}_{\mathsf{seed.col}}(w,\alpha) = \sum_{k\in[A]} \mathsf{coeff}_k(\alpha)\cdot C_{\mathsf{idx}_k(\alpha,\mathsf{icol}[\iota])}(w).$$

(Output product gates) There are $2^{r_{\mathsf{col}}}$ product gates. For each $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output gate is simply

$$(C_{\mathsf{norm}})_{\mathsf{seed.col}}(w,\alpha) = (\mathsf{Sum}_{\mathsf{seed.col}}(w,\alpha))^d.$$

Recall that this circuit C_{norm} has parameters as follows:

- the number of gates in each layer: $\ell_{\mathscr{C}} = \ell$, $\ell_{\mathsf{Sum}} = 2^{r_{\mathsf{col}}}$, $\ell_{\mathsf{Prod}} = 2^{r_{\mathsf{col}}}$;
- the fan-in of the top Prod gates d;
- the fan-in A, coefficient sum U, and locality l of the linear sum layer.

We invoke Theorem 3.7 on the circuit C_{norm} and $M' := 2^{r_{\mathsf{row}}}$ inputs $\{(w_{\mathsf{irow}[\iota]}, \alpha_{\mathsf{irow}[\iota]})\}_{\mathsf{seed.row}}$. We obtain an estimation $\mathsf{EST}_{\mathsf{norm}} = \mathsf{EST}_{\mathsf{norm}}(\mathsf{seed.shared}, \iota)$ where

$$\left|\mathsf{EST}_{\mathsf{norm}} - \|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d\right| \le \eta\cdot U^d.$$

If $\mathsf{EST}_{\mathsf{norm}} > 1+\eta\cdot U^d$, then we reject the input. Otherwise, we proceed to verify that π^{Real} and π^{Bool} are close. Consider the polynomial $P(z) := z^d(1-z)^d$, which intuitively measures how close z is to Boolean. We will estimate

$$\mathop{\mathbb{E}}_{i\leftarrow[\hat{H}_{\mathsf{proof}}],j\leftarrow[W_{\mathsf{proof}}]}\left[P(\pi^{\mathsf{Real}}_{i,j})\right]. \tag{22}$$

Similarly, we estimate Eq. (22) by building a $\mathsf{Prod}_{2d}\circ\mathsf{Sum}\circ\mathscr{C}$ circuit C_{diff}.

Circuit C_{diff}

(Inputs and bottom circuits) The inputs and bottom circuits of C_{diff} are exactly the same as C_{norm}.

(Intermediate linear sum gates) There are $2W_{\mathsf{proof}}$ intermediate linear sum gates. Let $j\in[W_{\mathsf{proof}}]$, then the $2j$-th linear sum gate computes $(\pi^{\mathsf{Real}})_j$, and the $(2j+1)$-th one computes $1-(\pi^{\mathsf{Real}})_j$. That is,

$$\mathsf{Sum}_{2j}(w,\alpha) = \sum_{k\in[A]}\mathsf{coeff}_k(\alpha)\cdot C_{\mathsf{idx}_k(\alpha,j)}(w);\quad \mathsf{Sum}_{2j+1}(w,\alpha) = 1-\mathsf{Sum}_{2j}(w,\alpha).$$

Implementation of the linear sum layer: Since we did not allow $\mathsf{coeff}_k(\alpha)$ to depend on i (the output index in $[2W_{\mathsf{proof}}]$), we need to be careful when implementing the linear sum layer. The fan-in of this layer will be $2A+1$ (instead of A). We identify $[2A+1]$ with the disjoint union of $[A]\times\{0,1\}$ and $\{\star\}$ (where $\star$ denotes the constant term 1 in $\mathsf{Sum}_{2j+1}(w,\alpha)$). Let idx' and coeff' be the idx and coeff functions of the intermediate linear sum gates of C_{diff}:

(Function $\mathsf{idx}'_k(\alpha,i)$) We write $i = 2j+b$ where $j\in[W_{\mathsf{proof}}]$ and $b\in\{0,1\}$. If $k=(k',b')\in[A]\times\{0,1\}$, then $\mathsf{idx}'_k(\alpha,i)$ returns $\mathsf{idx}_{k'}(\alpha,j)$ if $b=b'$ and returns ZERO if $b\neq b'$. If $k=\star$ then $\mathsf{idx}'_k(\alpha,i)$ returns ZERO if $b=0$ and returns ONE if $b=1$.

(Function $\mathsf{coeff}'_k(\alpha)$) If $k=(k',b')\in[A]\times\{0,1\}$, then $\mathsf{coeff}'_k(\alpha) = (-1)^{b'}\cdot\mathsf{coeff}_{k'}(\alpha)$. If $k=\star$ then $\mathsf{coeff}'_k(\alpha)=1$.

The locality of $(\mathsf{idx}',\mathsf{coeff}')$ is still l. The coefficient sum becomes $2U+1$.

(Output product gates) There are W_{proof} product gates. For each $j \in [W_{\mathsf{proof}}]$, the j-th output gate is

$$C_{\mathsf{diff}}(w, \alpha) = (\mathsf{Sum}_{2j}(w, \alpha) \cdot \mathsf{Sum}_{2j+1}(w, \alpha))^d.$$

The parameters of the circuit C_{diff} are as follows:

- the number of gates in each layer: $\ell_{\mathscr{C}} = \ell$, $\ell_{\mathsf{Sum}} = 2W_{\mathsf{proof}}$, $\ell_{\mathsf{Prod}} = W_{\mathsf{proof}}$;
- the fan-in of the top Prod gates $2d$;
- the fan-in $2A + 1$, coefficient sum $2U + 1$, and locality l of the linear sum layer.

We invoke Theorem 3.7 on the circuit C_{diff} and the set of $M' := \hat{H}_{\mathsf{proof}}$ inputs $\{(w_i, \alpha_i)\}_{i \in [\hat{H}_{\mathsf{proof}}]}$, and obtain an estimation $\mathsf{EST}_{\mathsf{diff}}$ where

$$|\mathsf{EST}_{\mathsf{diff}} - (22)| \le \eta \cdot (2U + 1)^{2d}.$$

We then accept if and only if $\mathsf{EST}_{\mathsf{diff}} \le 2^d \cdot \delta + \eta(2U + 1)^{2d}$.

Complexity. Our algorithm calls the algorithm in Theorem 3.7 as a subroutine on the circuits C_{norm} and C_{diff}. It is easy to see that $A^d(2^{dl} + 2^{r_{\mathsf{row}}}/2^{r_{\mathsf{col}}}) \cdot O(T^{\mathrm{alg}})$ time is spent on each C_{norm}. Similarly, it takes $(2A + 1)^{2d}(2^{2dl} + \hat{H}_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) \cdot (W_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) \cdot O(T^{\mathrm{alg}})$ time to process C_{diff}. It follows that our algorithm runs in deterministic time

$$\begin{aligned}
&O(2^{r_{\mathsf{shared}}}) \cdot A^d(2^{dl} + 2^{r_{\mathsf{row}}}/2^{r_{\mathsf{col}}}) \cdot O(T^{\mathrm{alg}}) + (2A + 1)^{2d}(2^{2dl} + \hat{H}_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) \cdot (W_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) \cdot O(T^{\mathrm{alg}}) \\
&= O((3A)^{2d} T^{\mathrm{alg}}) \cdot \left(2^{dl + r_{\mathsf{shared}}} + 2^{r - 2r_{\mathsf{col}}} + 2^{2dl} \cdot W_{\mathsf{proof}}/2^{r_{\mathsf{col}}} + \hat{H}_{\mathsf{proof}} \cdot W_{\mathsf{proof}}/2^{2r_{\mathsf{col}}}\right) \\
&= O((3A)^{2d} T^{\mathrm{alg}}) \cdot \left(2^{2dl + r_{\mathsf{shared}}} + T \log^{O(m)} T/2^{2r_{\mathsf{col}}}\right).
\end{aligned}$$

(Recall that since $\log(W_{\mathsf{proof}}/2^{r_{\mathsf{col}}}) = (5/m) \log T$, $r_{\mathsf{shared}} \ge (10/m) \log T$, we have $W_{\mathsf{proof}}/2^{r_{\mathsf{col}}} \le 2^{r_{\mathsf{shared}}}$. Also, from $r := \log T + O(\log \log T + m \log m)$ and $m = \Theta(\log n/\delta)$ we know that $2^{r - 2r_{\mathsf{col}}} \le T \log^{O(m)} T/2^{2r_{\mathsf{col}}}$.)

Now it suffices to prove the completeness and soundness requirements. Before that, we need the following fact regarding the polynomial P. Let $z \in \mathbb{R}$, $d_{\mathsf{bin}}(z)$ be the distance between z to the closest Boolean value; namely $d_{\mathsf{bin}}(z) := \min\{|z|, |1 - z|\}$.

Fact A.1. *For every $z \in \mathbb{R}$, $d_{\mathsf{bin}}(z)^d \cdot 2^{-d} \le P(z) \le d_{\mathsf{bin}}(z)^d \cdot (1 + d_{\mathsf{bin}}(z))^d$.* ◇

Completeness. Suppose that for every $i \in [\hat{H}_{\mathsf{proof}}]$ and $j \in [W_{\mathsf{proof}}]$, $\pi^{\mathsf{Real}}_{i,j} \in [0, 1]$. Then for every $\iota \in [q]$ and $\mathsf{seed.shared} \in \{0, 1\}^{r_{\mathsf{shared}}}$, $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d \le 1$, and thus $\mathsf{EST}_{\mathsf{norm}} \le 1 + \eta \cdot U^d$.

Suppose in addition that for every $i \in [\hat{H}_{\mathsf{proof}}]$, $\|\pi^{\mathsf{Real}}_i - \pi^{\mathsf{Bool}}_i\|_1 \le \delta$. Then:

$$\begin{aligned}
(22) &\le \mathop{\mathbb{E}}_{i,j}\left[d_{\mathsf{bin}}(\pi^{\mathsf{Real}}_{i,j})^d \cdot (1 + d_{\mathsf{bin}}(\pi^{\mathsf{Real}}_{i,j}))^d\right] \\
&\le 2^d \cdot \mathop{\mathbb{E}}_{i,j}\left[d_{\mathsf{bin}}(\pi^{\mathsf{Real}}_{i,j})^d\right] \\
&\le 2^d \cdot \delta.
\end{aligned}$$

Therefore, $\mathsf{EST}_{\mathsf{diff}} \le 2^d \cdot \delta + \eta(2U + 1)^{2d}$ and our algorithm accepts.

Soundness. Suppose our algorithm accepts.

1. For every $\mathsf{seed.shared}$ and ι, we have $\mathsf{EST}_{\mathsf{norm}} \le 1+\eta\cdot U^d$ and thus $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d \le 1+2\eta\cdot U^d$.
2. We have (22) $\le 2^d\cdot\delta + 2\eta(2U+1)^{2d}$ and

$$\begin{aligned}\mathop{\mathbb{E}}_{i\leftarrow[\hat{H}_{\mathsf{proof}}],j\leftarrow[W_{\mathsf{proof}}]}\left[|\pi^{\mathsf{Real}}_{i,j}-\pi^{\mathsf{Bool}}_{i,j}|^d\right] &= \mathop{\mathbb{E}}_{i,j}\left[d_{\mathsf{bin}}(\pi^{\mathsf{Real}}_{i,j})^d\right]\\ &\le 2^d\cdot(22) && \text{(Fact A.1)}\\ &\le 4^d\cdot\delta+2^{d+1}\eta(2U+1)^{2d}. && \square\end{aligned}$$

Lemma 4.6. *Let $\mathscr{C}$ be a typical circuit class and $d\ge 2$ be an even number. Suppose there is an algorithm that takes as inputs a list of $2^{r_{\mathsf{col}}}$ $\mathsf{AND}_{2d}\circ\mathscr{C}$ circuits $\{C_i\}$ and a list of $2^{r_{\mathsf{col}}}$ inputs $\{x_j\}$ of length $n\cdot\mathrm{polylog}(\ell)$, runs in deterministic T^{alg} time, and estimates the following quantity with additive error η:*

$$\Pr_{i,j\leftarrow[2^{r_{\mathsf{col}}}]}[C_i(x_j)].$$

Then there is an algorithm that takes the strings $w_1,w_2,\ldots,w_\ell$, circuits $(C_1,C_2,\ldots,C_{\hat{H}_{\mathsf{proof}}})$, and $(\alpha_1,\alpha_2,\ldots,\alpha_{\hat{H}_{\mathsf{proof}}})$ as inputs, runs in deterministic $O((3A)^{2d}T^{\mathrm{alg}})\cdot(2^{2dl+r_{\mathsf{shared}}}+T\log^{O(m)}T/2^{2r_{\mathsf{col}}}))$ time, and satisfies the following:

(Completeness) *If for every $i\in[\hat{H}_{\mathsf{proof}}]$, it holds that (1) for every $j\in[W_{\mathsf{proof}}]$, $\pi^{\mathsf{Real}}_{i,j}\in[0,1]$; (2) $\|\pi^{\mathsf{Real}}_i-\pi^{\mathsf{Bool}}_i\|_1\le\delta$, then the algorithm accepts.*

(Soundness) *If the algorithm accepts, then it holds that*

1. *for every $\mathsf{seed.shared}\in\{0,1\}^{r_{\mathsf{shared}}}$ and $\iota\in[q]$, $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d\le 1+2\eta\cdot U^d$;*
2. *$\mathbb{E}_{i\leftarrow[\hat{H}_{\mathsf{proof}}],j\leftarrow[W_{\mathsf{proof}}]}\left[|\pi^{\mathsf{Real}}_{i,j}-\pi^{\mathsf{Bool}}_{i,j}|^d\right]\le 4^d\cdot\delta+2^{d+1}\eta(2U+1)^{2d}$.*

Proof Sketch. We first estimate $\|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d$ for fixed $\mathsf{seed.shared}$ and ι. Recall that

$$\pi^{\mathsf{Real}}_{i,j}=\sum_{k\in[A]}\mathsf{coeff}_k(\alpha_i)\cdot C_i(w_{\mathsf{idx}_k(\alpha_i,j)}).$$

We build a $\mathsf{Prod}_d\circ\mathsf{Sum}$ circuit $C_{\mathsf{norm}}:=C_{\mathsf{norm}}(\mathsf{seed.shared},\iota)$ as follows.

Circuit C_{norm}

(Inputs) The input consists of (y,α) with the intended meaning that $y=(y_1,y_2,\ldots,y_\ell)$ where $y_i=C_{\mathsf{irow}[\iota]}(w_i)$, and $\alpha=\alpha_{\mathsf{irow}[\iota]}$.

(Linear sum gates) There are $2^{r_{\mathsf{col}}}$ linear sum gates. For each $\mathsf{seed.col}$,

$$\mathsf{Sum}_{\mathsf{seed.col}}(y,\alpha)=\sum_{k\in[A]}\mathsf{coeff}_k(\alpha)\cdot y_{\mathsf{idx}_k(\alpha,\mathsf{icol}[\iota])}.$$

(Output product gates) There are $2^{r_{\mathsf{col}}}$ product gates. For each $\mathsf{seed.col}$, the $\mathsf{seed.col}$-th output gate is simply

$$(C_{\mathsf{norm}})_{\mathsf{seed.col}}(y,\alpha)=(\mathsf{Sum}_{\mathsf{seed.col}}(y,\alpha))^d.$$

Recall that this circuit C_{norm} has parameters as follows:

- the number of gates in each layer: $\ell_{\mathsf{Sum}} = 2^{r_{\mathsf{col}}}$, $\ell_{\mathsf{Prod}} = 2^{r_{\mathsf{col}}}$;
- the fan-in of the top Prod gates d;
- the fan-in A, coefficient sum U, and locality l of the linear sum layer.

We invoke Lemma 4.5 on the circuit C_{norm}, strings $w_1, w_2, \ldots, w_\ell$, a list of $2^{r_{\mathsf{row}}}$ inputs $\{\alpha_{\mathsf{irow}[\iota]}\}_{\mathsf{seed.row}}$, and a list of $2^{r_{\mathsf{row}}}$ size-s $\mathscr{C}$ circuits $\{C_{\mathsf{irow}[\iota]}\}_{\mathsf{seed.row}}$. Here $\ell_{\mathscr{C}} = 1$. We thus obtain an estimation $\mathsf{EST}_{\mathsf{norm}} = \mathsf{EST}_{\mathsf{norm}}(\mathsf{seed.shared}, \iota)$ where

$$\left|\mathsf{EST}_{\mathsf{norm}} - \|f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_d^d\right| \leq \eta \cdot U^d.$$

If $\mathsf{EST}_{\mathsf{norm}} > 1 + \eta \cdot U^d$, then we reject the input. Otherwise, we proceed to verify that π^{Real} and π^{Bool} are close. Consider the polynomial $P(z) := z^d(1-z)^d$. we will estimate

$$\mathop{\mathbb{E}}_{i \leftarrow [\hat{H}_{\mathsf{proof}}], j \leftarrow [W_{\mathsf{proof}}]} \left[P(\pi^{\mathsf{Real}}_{i,j})\right]. \tag{23}$$

Similarly, we estimate Eq. (23) by building a $\mathsf{Prod}_{2d} \circ \mathsf{Sum}$ circuit C_{diff}.

Circuit C_{diff}

(Inputs) The inputs are exactly the same as C_{norm}.

(Linear sum gates) There are $2W_{\mathsf{proof}}$ linear sum gates. Let $j \in [W_{\mathsf{proof}}]$, then the $2j$-th linear sum gate computes $(\pi^{\mathsf{Real}})_j$, and the $(2j+1)$-th one computes $1 - (\pi^{\mathsf{Real}})_j$. That is,

$$\mathsf{Sum}_{2j}(y, \alpha) = \sum_{k \in [A]} \mathsf{coeff}_k(\alpha) \cdot y_{\mathsf{idx}_k(\alpha, j)}; \quad \mathsf{Sum}_{2j+1}(y, \alpha) = 1 - \mathsf{Sum}_{2j}(y, \alpha).$$

The implementation of the linear sum layer is the same as in Lemma 3.8, and we omit it here.

(Output product gates) There are W_{proof} product gates. For each $j \in [W_{\mathsf{proof}}]$, the j-th output gate is

$$C_{\mathsf{diff}}(y, \alpha) = (\mathsf{Sum}_{2j}(y, \alpha) \cdot \mathsf{Sum}_{2j+1}(y, \alpha))^d.$$

The parameters of the circuit C_{diff} are as follows:

- the number of gates in each layer: $\ell_{\mathsf{Sum}} = 2W_{\mathsf{proof}}$, $\ell_{\mathsf{Prod}} = W_{\mathsf{proof}}$;
- the fan-in of the top Prod gates $2d$;
- the fan-in $2A + 1$, coefficient sum $2U + 1$, and locality l of the linear sum layer.

We invoke Lemma 4.5 on the circuit C_{diff}, strings $w_1, w_2, \ldots, w_\ell$, a list of $\hat{H}_{\mathsf{proof}}$ inputs $\{\alpha_i\}$, and a list of $\hat{H}_{\mathsf{proof}}$ size-s $\mathscr{C}$ circuits $\{C_i\}$. Here $\ell_{\mathscr{C}} = 1$. We obtain an estimation $\mathsf{EST}_{\mathsf{diff}}$ where

$$|\mathsf{EST}_{\mathsf{diff}} - (23)| \leq \eta \cdot (2U+1)^{2d}.$$

We accept if and only if $\mathsf{EST}_{\mathsf{diff}} \leq 2^d \cdot \delta + \eta(2U+1)^{2d}$.

The correctness and complexity are analysed the same as in Lemma 3.8, so we omit it here. ◇

A.3 Proof of Claim 3.9

We need the following technical lemma (see [CW19b, Lemma 28] and [RSW22, Lemma 4.9]):

Lemma A.2. *Let $d \geq 2$ be an integer, $f_1, f_2, \ldots, f_d, g_1, g_2, \ldots, g_d : [N] \to \mathbb{R}$ be functions. For all $i \in [d]$, suppose that $\|f_i\|_d \leq 1$, and define $\varepsilon := \sum_{i=1}^d \|f_i - g_i\|_d$. Then*

$$\left|\mathop{\mathbb{E}}_{x \leftarrow [N]}\left[\prod_{i=1}^d f_i(x) - \prod_{i=1}^d g_i(x)\right]\right| \leq (1+\varepsilon)^{d-1} \cdot \varepsilon.$$

The above lemma is a consequence of the following generalisation of Hölder's inequality:

Fact A.3. *Let $f_1, f_2, \ldots, f_d : [N] \to \mathbb{R}$ be functions, $f : [N] \to \mathbb{R}$ be their product, i.e., $f(x) = \prod_{i=1}^{d} f_i(x)$. Then $\|f\|_1 \le \prod_{i=1}^{d} \|f_i\|_d$.*

Proof of Lemma A.2. Let $\varepsilon_i := \|f_i - g_i\|_d$, then $\varepsilon = \sum_{i=1}^{d} \varepsilon_i$. Define

$$\mathsf{Hyb}_i := \mathop{\mathbb{E}}_{x \leftarrow [N]} \left[\prod_{j=1}^{i} f_j(x) \cdot \prod_{j=i+1}^{d} g_j(x) \right].$$

Then, for every $1 \le i \le d$,

$$\begin{aligned}
|\mathsf{Hyb}_i - \mathsf{Hyb}_{i-1}| &\le \mathop{\mathbb{E}}_{x \leftarrow [N]} \left[\left| \prod_{j=1}^{i-1} f_j(x) \cdot \prod_{j=i+1}^{d} g_j(x) \cdot (f_i(x) - g_i(x)) \right| \right] \\
&\le \prod_{j=1}^{i-1} \|f_j\|_d \cdot \prod_{j=i+1}^{d} \|g_j\|_d \cdot \|f_i - g_i\|_d \qquad \text{(Fact A.3)} \\
&\le \prod_{j=2}^{d} (1 + \varepsilon_j) \cdot \varepsilon_i \\
&\le (1+\varepsilon)^{d-1} \cdot \varepsilon_i.
\end{aligned}$$

It follows that

$$\left| \mathop{\mathbb{E}}_{x \leftarrow [N]} \left[\prod_{i=1}^{d} f_i(x) - \prod_{i=1}^{d} g_i(x) \right] \right| = |\mathsf{Hyb}_d - \mathsf{Hyb}_0| \le \sum_{i=1}^{d} |\mathsf{Hyb}_i - \mathsf{Hyb}_{i-1}| \le (1+\varepsilon)^{d-1} \cdot \varepsilon. \qquad \square$$

Recall that for $S, S' \subseteq [q]$, we define

$$\begin{aligned}
p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') &:= \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row} \leftarrow \{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col} \leftarrow \{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{\iota \in S} (\mathsf{row}_\iota^{\mathsf{Bool}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} pc_\iota \right], \\
p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S') &:= \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row} \leftarrow \{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col} \leftarrow \{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{\iota \in S} (\mathsf{row}_\iota^{\mathsf{Real}})_{\mathsf{icol}[\iota]} \cdot \prod_{\iota \in S'} pc_\iota \right], \\
\delta_{\mathsf{seed.shared}} &:= \sum_{\iota : \mathsf{itype}[\iota] = \mathsf{proof}} \|f^{\mathsf{Bool}}_{\mathsf{seed.shared},\iota} - f^{\mathsf{Real}}_{\mathsf{seed.shared},\iota}\|_{2q}.
\end{aligned}$$

Claim 3.9. *For every $S, S' \subseteq [q]$,*

$$|p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') - p_{\mathsf{acc}}^{\mathsf{Real}}(\mathsf{seed.shared}, S, S')| \le (1 + \delta_{\mathsf{seed.shared}})^{2q-1} \cdot \delta_{\mathsf{seed.shared}}.$$

Proof. Define the following $2(|S| + |S'|)$ functions f_i^{Bool}, g_j^{Bool}, f_i^{Real}, g_j^{Real}, where $i \in S$ and $j \in S'$. Each function takes $(\mathsf{seed.row}, \mathsf{seed.col})$ as inputs, and:

$$f_i^{\mathsf{Bool}} := (\mathsf{row}_i^{\mathsf{Bool}})_{\mathsf{icol}[i]}; \quad f_i^{\mathsf{Real}} := (\mathsf{row}_i^{\mathsf{Real}})_{\mathsf{icol}[i]}; \quad \text{and} \quad g_j^{\mathsf{Bool}} = g_j^{\mathsf{Real}} := pc_j.$$

(Note: for convenience, we omit the input $(\mathsf{seed.row}, \mathsf{seed.col})$.) It follows that $\|f_i^{\mathsf{Bool}}\|_{2q}, \|g_j^{\mathsf{Bool}}\|_{2q} \le 1$; for every $j \in S'$, $\|g_i^{\mathsf{Bool}} - g_i^{\mathsf{Real}}\|_{2q} = 0$; and for every $i \in S$,

$$\|f_i^{\mathsf{Bool}} - f_i^{\mathsf{Real}}\|_{2q} = \begin{cases} 0 & \text{if } \mathsf{itype}[i] = \mathsf{input}; \\ \|f^{\mathsf{Bool}}_{\mathsf{seed.shared},i} - f^{\mathsf{Real}}_{\mathsf{seed.shared},i}\|_{2q} & \text{if } \mathsf{itype}[i] = \mathsf{proof}. \end{cases}$$

Therefore, by Lemma A.2,

$$\begin{aligned}
&|p_{\mathsf{acc}}(\mathsf{seed.shared}, S, S') - p^{\mathsf{Real}}_{\mathsf{acc}}(\mathsf{seed.shared}, S, S')| \\
=&\left| \mathop{\mathbb{E}}_{\substack{\mathsf{seed.row}\leftarrow\{0,1\}^{r_{\mathsf{row}}} \\ \mathsf{seed.col}\leftarrow\{0,1\}^{r_{\mathsf{col}}}}} \left[\prod_{i\in S} f_i^{\mathsf{Bool}} \cdot \prod_{j\in S'} g_j^{\mathsf{Bool}} - \prod_{i\in S} f_i^{\mathsf{Real}} \cdot \prod_{j\in S'} g_j^{\mathsf{Real}} \right] \right| \\
\le&(1+\delta_{\mathsf{seed.shared}})^{2q-1} \cdot \delta_{\mathsf{seed.shared}}.
\end{aligned}$$

□

A.4 Proof of Lemma 3.10

Lemma 3.10. *Let $f : [N] \times [q] \to \mathbb{R}_{\ge 0}$ be a function and $d \ge 1$ be a constant. Suppose that*

1. *for every $s \in [N]$ and $i \in [q]$, $f(s,i) \le \alpha$ (where $\alpha \ge 1$);*
2. *$\mathbb{E}_{s,i}[f(s,i)^d] \le \delta$.*

Let $f(s) := \sum_{i\in[q]} f(s,i)$. Then

$$\mathop{\mathbb{E}}_{s}[(1+f(s))^{d-1} \cdot f(s)] \le q\delta^{1/d}(2q\alpha)^{d-1}.$$

Proof. By Jensen's inequality,

$$\mathop{\mathbb{E}}_{s}[f(s)] = q \mathop{\mathbb{E}}_{s,i}[f(s,i)] \le q\delta^{1/d}.$$

It follows that for every $k \ge 1$,

$$\mathop{\mathbb{E}}_{s}[f(s)^k] \le \mathop{\mathbb{E}}_{s}[f(s)] \cdot \max_s\{f(s)\}^{k-1} \le q\delta^{1/d} \cdot (q\alpha)^{k-1}.$$

Finally, we have

$$\mathop{\mathbb{E}}_{s}[(1+f(s))^{d-1} \cdot f(s)] = \sum_{i=0}^{d-1} \binom{d-1}{i} \cdot \mathop{\mathbb{E}}_{s}[f(s)^{i+1}] \le q\delta^{1/d}(2q\alpha)^{d-1}.$$

□

A.5 An XOR Lemma in [CLW20]

Theorem 2.5. *Let $N \in \mathbb{N}$, $0 < \varepsilon, \delta < 1/10$, $k := O(\log(1/\varepsilon)/\delta)$, $\tilde{N} := N^k$, and $a := O(\log^2 N/(\varepsilon\delta)^2)$. There is an algorithm $\mathsf{Amp} : \{0,1\}^N \to \{0,1\}^{\tilde{N}}$ computable in deterministic $\mathrm{poly}(\tilde{N})$ time, and a linear sum circuit $C : \{0,1\}^{\tilde{N}} \times \{0,1\}^a \to \mathbb{R}^N$ such that the following hold.*

(List-decoding) *For every string $\tilde{f} \in \{0,1\}^{\tilde{N}}$ that is $(1/2-\varepsilon)$-close to $\mathsf{Amp}(f)$ for some hidden string f, there is an advice $\alpha \in \{0,1\}^a$, such that (1) for every $i \in [N]$, $C(\tilde{f},\alpha)_i \in [0,1]$; and (2) $\|C(\tilde{f},\alpha) - f\|_1 \le \delta$.*

(Complexity) *The fan-in, coefficient sum, and locality of C are at most $O(\log N/(\varepsilon\delta)^2)$, $O(1/\varepsilon)$, and $\log \tilde{N}$ respectively.*

The lemma is implied by the XOR lemma in [CLW20]. For simplicity, we identify a string f of length N and a Boolean function $f : [N] \to \{0,1\}$, where $f(x)$ outputs the x-th bit of f. For a string $f \in \{0,1\}^N$, denote $f^{\oplus k}$ to be the following string of length N^k. For each $(x_1, x_2, \ldots, x_k) \in [N]^k$, we have

$$f^{\oplus k}(x_1, x_2, \ldots, x_k) := \bigoplus_{i=1}^{k} f(x_i).$$

And we simply let $\mathsf{Amp}(f) := f^{\oplus k}$.

The decoder. For a length-k vector $\vec{v}^{\perp} \in ([N] \cup \{\perp\})^k$ and $i \in [N]$, let $\vec{v}^{i}$ denote the vector where each $\perp$ in $\vec{v}^{\perp}$ is replaced by i. (In the decoder, we will only need the case where each $\vec{v}^{\perp}$ contains exactly one $\perp$, so $\vec{v}^{i}$ simply replaces that single $\perp$ by i.)

Let $\tilde{f} : [N]^k \to \{0,1\}$ be a codeword (treated as a Boolean function). For $A' := O(\log N/(\varepsilon\delta)^2)$ and $r := \frac{(2+\delta)\varepsilon}{1-\delta}$, our decoder will take a list of vectors $\vec{v}_1^{\perp}, \vec{v}_2^{\perp}, \ldots, \vec{v}_{A'}^{\perp} \in ([N] \cup \{\perp\})^k$ and a list of signs $\sigma_1, \sigma_2, \ldots, \sigma_{A'} \in \{0,1\}$ as advice. Intuitively, $\vec{v}_i^{\perp}$ denotes a segment of $\tilde{f}$ that has noticeable correlation with f, and σ_i denotes whether the correlation is positive or negative; our linear sum decoder uses the average of $\tilde{f}(\vec{v}_j^{i}) \oplus \sigma_j$ as a prediction of f_i.

More formally, given an input $i \in [N]$, the decoder outputs [30]

$$\mathsf{dec}(\tilde{f})_i := \frac{1}{r} \mathop{\mathbb{E}}_{j \leftarrow [A']} \Big[(\tilde{f}(\vec{v}_j^{i}) \oplus \sigma_j) - 1/2 \Big] + 1/2. \tag{24}$$

Correctness. We establish the correctness of this decoder by the following lemma.

Lemma A.4. *Let $k \geq 1$, $\delta \in (0, 1/10)$, $\varepsilon := (1-\delta)^{k-1}(1/2 - \delta)$, and $A' := O(\log N/(\varepsilon\delta)^2)$. For every string $\tilde{f} \in \{0,1\}^{N^k}$ that is $(1/2 - \varepsilon)$-close to $f^{\oplus k}$ for some hidden string $f \in \{0,1\}^N$, there is a list of A' vectors $\vec{v}_1^{\perp}, \vec{v}_2^{\perp}, \ldots, \vec{v}_{A'}^{\perp} \subseteq ([N] \cup \{\perp\})^k$, and a list of signs $\sigma_1, \sigma_2, \ldots, \sigma_{A'} \in \{0,1\}$, such that (1) for every $i \in [N]$, $\mathsf{dec}(\tilde{f})_i \in [0,1]$; and (2) $\|\mathsf{dec}(\tilde{f}) - f\|_1 \leq \delta$.*

Proof. We use induction on k. Suppose $k = 1$, then one can verify by direct calculation that the lemma holds by setting $\vec{v}_1^{\perp} = (\perp)$ and $\sigma_1 = 0$. Now suppose $k > 1$ and the lemma holds for $k-1$.

Fix $i \in [N]$ and let $\vec{v}^{\perp} \in ([N] \cup \{\perp\})^k$ denote some vector whose first coordinate is $\perp$ and other coordinates are from $[N]$. Think of every coordinate of $\vec{v}^{\perp}$, except the first, is drawn independently and uniformly from $[N]$. Define

$$p_i := \Pr_{\vec{v}^{\perp} \leftarrow \{\perp\} \times [N]^{k-1}} \Big[\tilde{f}(\vec{v}^{i}) = f^{\oplus k}(\vec{v}^{i}) \Big].$$

Case I: Suppose there is some $i_0 \in [N]$ such that $|p_{i_0} - 1/2| > \varepsilon/(1-\delta)$. Let $b \in \{0,1\}$ be a bit, consider the sub-string $\tilde{f}' \in \{0,1\}^{N^{k-1}}$ such that $\tilde{f}'(\vec{v}^{\perp}) = \tilde{f}(\vec{v}^{i_0}) \oplus b$. Then, for some $b \in \{0,1\}$, $\tilde{f}'$ is $(1/2 - \varepsilon/(1-\delta))$-close to $f^{\oplus(k-1)}$.

By the induction hypothesis, there is a list of A' vectors $\vec{u}_1^{\perp}, \ldots, \vec{u}_{A'}^{\perp} \subseteq ([N-1] \cup \{\perp\}^k)$ and a list of signs $\sigma'_1, \ldots, \sigma'_{A'} \in \{0,1\}$ such that the vector dec' satisfies the conclusion of the lemma, where

$$\mathsf{dec}'_i := \frac{1}{r} \mathop{\mathbb{E}}_{j \leftarrow [A']} \Big[(\tilde{f}'(\vec{u}_j^{i}) \oplus \sigma'_j) - 1/2 \Big] + 1/2.$$

For each j, let $\vec{v}_j^{\perp}$ be the concatenation of i_0 and $\vec{u}_j^{\perp}$, and let $\sigma_j = \sigma'_j \oplus b$. We have that $\mathsf{dec}(\tilde{f})_i$ is exactly dec'_i and we are done.

[30] Eq. (24) is perhaps easier to understand when we change the basis from $\{0,1\}$ to $\{1,-1\}$; we choose the basis $\{0,1\}$ only to be consistent with other parts of this paper. When we change the basis to $\{1,-1\}$, XOR becomes multiplication and the assertion "$a = b$" becomes simply $a \cdot b$. Thus Eq. (24) becomes

$$\mathsf{dec}(\tilde{f})_i = \frac{1}{r} \mathop{\mathbb{E}}_{j \leftarrow [A']} \Big[\tilde{f}(\vec{v}_j^{i}) \cdot \sigma_j \Big],$$

which is simply the average of all A' predictions, amplified by a factor of $1/r$.

Case II: Suppose for every $i \in [N]$, we have $|p_i - 1/2| \le \varepsilon/(1-\delta)$. Note that, since $\tilde{f}$ is $(1/2-\varepsilon)$-close to $f^{\oplus k}$, we have

$$\mathop{\mathbb{E}}_{i \leftarrow [N]}[p_i] \ge 1/2 + \varepsilon.$$

We sample each $\vec{v}_j^{\perp} \leftarrow \{\perp\} \times [N]^{k-1}$ independently at random. Let

$$\tilde{p}_i := \Pr_{j \leftarrow [A']}\Big[\tilde{f}(\vec{v}_j^{\,i}) = f^{\oplus k}(\vec{v}_j^{\,i})\Big].$$

Let $\eta := \frac{\varepsilon\delta}{2(1-\delta)}$. By a Chernoff bound, w.p. $1 - Ne^{-2\eta^2 t} > 0$, for every $i \in [N]$, we have $|p_i - \tilde{p}_i| \le \eta$. Let $\sigma_j := f^{\oplus(k-1)}((\vec{v}_j^{\perp})_{2\sim k}) = \bigoplus_{l=2}^{k} f((\vec{v}_j^{\perp})_l)$, then we have

$$\tilde{p}_i = \Pr_{j \leftarrow [t]}\Big[f_i = \tilde{f}(\vec{v}_j^{\,i}) \oplus \sigma_j\Big].$$

Note that

$$\begin{aligned}
\mathsf{dec}(\tilde{f})_i &= \frac{1}{r}(\tilde{p}_i \cdot (f_i - 1/2) + (1-\tilde{p}_i)\cdot(1/2 - f_i)) + 1/2 \\
&= \frac{(f_i - 1/2)(2\tilde{p}_i - 1)}{r} + 1/2.
\end{aligned}$$

Claim A.5. *For every $i \in [N]$, $\mathsf{dec}(\tilde{f})_i \in [0,1]$.*

Proof.

$$\begin{aligned}
|\mathsf{dec}(\tilde{f})_i - 1/2| &= \frac{1}{r}|(f_i - 1/2)(2\tilde{p}_i - 1)| \\
&= \frac{1}{r}|\tilde{p}_i - 1/2| \\
&\le \frac{1-\delta}{(2+\delta)\varepsilon}\left(\frac{\varepsilon}{1-\delta} + \eta\right) \\
&= 1/2.
\end{aligned}$$

◇

Claim A.6. $\|\mathsf{dec}(\tilde{f}) - f\|_1 = \mathbb{E}_{i \leftarrow [N]}[|\mathsf{dec}(\tilde{f})_i - f_i|] \le \delta$.

Proof.

$$\begin{aligned}
&\mathop{\mathbb{E}}_{i \leftarrow [N]}[|\mathsf{dec}(\tilde{f})_i - f_i|] \\
&= \mathop{\mathbb{E}}_{i \leftarrow [N]}\left[\left|\frac{(f_i - 1/2)(2\tilde{p}_i - 1)}{r} + 1/2 - f_i\right|\right] \\
&= \mathop{\mathbb{E}}_{i \leftarrow [N]}\left[\left|\frac{\tilde{p}_i - 1/2}{r} - 1/2\right|\right] \\
&= 1/2 - \frac{\mathbb{E}_{i \leftarrow [N]}[\tilde{p}_i] - 1/2}{r} \qquad (25) \\
&\le 1/2 - \frac{\varepsilon - \eta}{r} \le \delta,
\end{aligned}$$

where Eq. (25) is because we have shown in the proof of Claim A.5 that $\frac{1}{r}|\tilde{p}_i - 1/2| \le 1/2$ for every i. ◇

Combining Claim A.5 and A.6, the lemma is proved. □

Complexity. It remains to determine the complexity of the decoder defined in Eq. (24). The advice string α contains the vectors $\vec{v}_1^{\perp}, \vec{v}_2^{\perp}, \ldots, \vec{v}_{A'}^{\perp}$ and the signs $\sigma_1, \sigma_2, \ldots, \sigma_{A'}$. It is clear that the fan-in is at most $A' + 1 = O(\log N/(\varepsilon\delta)^2)$. The coefficient sum is $O(1/r) = O(\varepsilon^{-1})$. Since the k-th term is

$$\mathsf{coeff}_k(\alpha) = (-1)^{\sigma_j}/(A'r), \quad \text{and} \quad \mathsf{idx}_k(\alpha, i) = \vec{v}_k^{\,i},$$

it follows that each term only depends on $\log \tilde{N}$ bits of α.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il