Rigid Matrices From Rectangular PCPs
or: Hard Claims Have Complex Proofs

Amey Bhangale [*][†]   Prahladh Harsha [‡][†]   Orr Paradise [§][†]   Avishay Tal [¶]

May 6, 2020

Abstract

We introduce a variant of PCPs, that we refer to as *rectangular* PCPs, wherein proofs are thought of as square matrices, and the random coins used by the verifier can be partitioned into two disjoint sets, one determining the *row* of each query and the other determining the *column*.

We construct PCPs that are *efficient*, *short*, *smooth* and (almost-)*rectangular*. As a key application, we show that proofs for hard languages in $\mathsf{NTIME}(2^n)$, when viewed as matrices, are rigid infinitely often. This strengthens and considerably simplifies a recent result of Alman and Chen [FOCS, 2019] constructing explicit rigid matrices in FNP. Namely, we prove the following theorem:

- There is a constant $\delta \in (0,1)$ such that there is an FNP-machine that, for infinitely many N, on input 1^N outputs $N \times N$ matrices with entries in $\mathbb{F}_2$ that are δN^2-far (in Hamming distance) from matrices of rank at most $2^{\log N/\Omega(\log \log N)}$.

Our construction of rectangular PCPs starts with an analysis of how randomness yields queries in the Reed–Muller-based outer PCP of Ben-Sasson, Goldreich, Harsha, Sudan and Vadhan [SICOMP, 2006; CCC, 2005]. We then show how to preserve rectangularity under PCP composition and a smoothness-inducing transformation. This warrants refined and stronger notions of rectangularity, which we prove for the outer PCP and its transforms.

1 Introduction

An $N \times N$ matrix with entries from a field $\mathbb{F}$ is said to be (Δ, ρ)-*rigid* if its Hamming distance from the set of $N \times N$ matrices of rank at most ρ is greater than Δ. In other words, an (Δ, ρ)-rigid matrix is a matrix that cannot be expressed as a sum of two matrices, $L + S$, where $\text{rank}(L) \leq \rho$ and S has at most Δ non-zero entries. For concreteness, this work focuses on rigidity with respect to the field $\mathbb{F}_2$.

Constructing rigid matrices has been a long-standing open problem in computational complexity theory since their introduction by Valiant more than four decades ago [Val77]. Valiant showed that for any $(N^{1+\varepsilon}, N/\log\log(N))$-rigid matrix, evaluating the corresponding linear transformation requires circuits of either super-linear size or super-logarithmic depth. Thus, an explicit construction of a such matrices gives explicit problems that cannot be solved in linear-size logarithmic-depth circuits.

Razborov [Raz89] (see also [Wun12]) considered the other end of the spectrum of parameters, in which the distance Δ is quite high but the rank ρ is much smaller; namely, $\Delta = \delta \cdot N^2$ for constant $\delta > 0$ and $\rho = 2^{(\log\log N)^{\omega(1)}}$. Razborov showed that strongly-explicit matrices[1] with these rigidity parameters imply a lower bound for the communication-complexity analog of the Polynomial Hierarchy, PH^{cc}.

In other words, while Valiant's regime focuses on very high rank ρ, Razborov's focuses on very high distance Δ. Achieving lower bounds for either PH^{cc} (via Razborov's reduction) or linear-size log-depth circuits (via Valiant's reduction) are two central long-standing open questions in complexity theory.

Despite a lot of effort, state of the art results on matrix rigidity fall short of solving both Valiant and Razborov's challenges. The current best poly-time constructions yields $(\frac{N^2}{\rho}\log(\frac{N}{\rho}), \rho)$-rigid matrices, for any parameter ρ (see Friedman [Fri93]; Shokrollahi, Spielman, and Stemann [SSS97]). Goldreich and Tal [GT18] gave a randomized poly-time algorithm that uses $O(N)$ random bits and produces an $N \times N$ matrix that is $(\frac{N^3}{\rho^2 \log N}, \rho)$-rigid for any parameter $\rho \geq \sqrt{N}$, with high probability.[2] Recent breakthrough results [AW17, DE19, DL19] showed that several long-standing candidate construction of explicit matrices, like the Hadamard matrix, or the FFT matrix, are less rigid than previously believed, and in particular do not meet Valiant's challenge.

Most previous attempts were of combinatorial or algebraic nature (see survey of [Lok09]). In contrast to these, a recent remarkable work of Alman and Chen [AC19] proposes a novel approach that uses ideas from complexity theory to construct rigid matrices in FNP:[3]

Theorem 1.1 ([AC19]). *There exists a constant $\delta \in (0,1)$ such that for all $\varepsilon \in (0,1)$, there is an FNP-machine that, for infinitely many N, on input 1^N outputs an $N \times N$ matrix that is $(\delta \cdot N^2, 2^{(\log N)^{1/4-\varepsilon}})$-rigid.*[4]

Their result still does not attain the rank bounds required for Valiant's lower bounds, yet it vastly improves the state of the art of explicit rigid matrix constructions. On Razborov's end, the construction indeed meets the required rigidity parameters (in fact, greatly exceeds them), but does not fulfill the requirement of super-explicitness. That said, they use a tensoring argument to obtain $N \times N$ matrices still within Razborov's rigidity parameters, in which each entry is computable in non-deterministic time $2^{(\log\log N)^{\omega(1)}}$. While this is not super-explicit, it is an exponential improvement over previous results.

The surprising construction of Alman and Chen is a tour-de-force that ties together seemingly unrelated areas of complexity theory. A key area is the theory of Probabilistically Checkable Proofs (PCPs). PCPs provide a format of rewriting classical NP-proofs that can be efficiently verified based only on a small amount of random queries into the rewritten proof. The PCP Theorem [AS98, ALMSS98] asserts that any NP-proof can be rewritten into a polynomially-longer PCP that can be verified using a constant number of queries.

---

[1]An $N \times N$ matrix A is strongly-explicit if, given $i, j \in N$, one can compute $A_{i,j}$ in $\mathsf{polylog}(N)$ time.

[2]Despite its "semi-explicitness", if this construction obtains Valiant's rigidity parameters, then lower bounds would be implied, since one can take the randomness to be part of the input, yielding a (related) explicit problem that has no linear-size logarithmic-depth circuits.

[3]The complexity class FNP is the function-problem extension of the decision-problem class NP. Formally, a relation $R(x, y)$ is in FNP if there exists a non-deterministic polynomial-time Turing machine M such that for any input x, $M(x)$ outputs y such $R(x, y) = 1$ or rejects if no such x exists.

[4]Their result is stated for $\mathsf{FP}^{\mathsf{NP}}$ but can be strengthened to FNP. More precisely, on infinitely many inputs 1^N that are accepted by the FNP-machine, any accepting path outputs a rigid matrix. Matrices obtained on different accepting paths may differ, but all of them are rigid. If one insists on outputting the same matrix on all accepting paths, then this can be done in $\mathsf{FP}^{\mathsf{NP}}$.

Alman and Chen make use of *efficient* and *short* PCPs for $\mathsf{NTIME}(2^n)$, as well as *smooth* PCPs. Momentarily, we too will make use of these properties, so let us give an informal description of these:

Efficient PCP: A PCP for $\mathsf{NTIME}(T(n))$ is said to be *efficient* if the running time of the PCP verifier is sub-linear (or even logarithmic) in the length of the original NP-proof (i.e., $T(n)$).

Short PCP: A PCP for $\mathsf{NTIME}(T(n))$ is said to be *short* if the length of the PCP is nearly linear (i.e., $T(N)^{1+o(1)}$) in the length of the original NP-proof (i.e., $T(n)$).

Smooth PCP: A PCP is said to be *smooth* if for each input, every proof location is equally likely to be queried by the PCP verifier.

The PCP Theorem had a remarkable impact on our understanding of the hardness of approximation of several combinatorial problems (see, e.g., a survey [Tre13]). Parallel to this line of work, Babai, Fortnow, Levin and Szegedy [BFLS91] initiated a long sequence of works [BSVW03, BGHSV06, BGHSV05, BS08, Din07, MR08, Mie09, BV14, Par20] that prove that there exist efficient, short and smooth PCPs for $\mathsf{NTIME}(2^n)$. Alman and Chen's construction makes use of these PCPs, as well as the non-deterministic time-hierarchy theorem [Zák83] and a fast (i.e., faster than $N^2/\mathsf{polylog}(N)$ time for $N \times N$ matrices) algorithm for counting the number of ones in a low-rank matrix [CW16].

1.1 Our results

Our work arises from asking if there exist PCPs with additional "nice" properties that can strengthen the above construction due to Alman and Chen. We answer this question in the affirmative, by (1) introducing a new variant of PCPs that we refer to as *rectangular PCPs*, (2) constructing efficient, short and smooth rectangular PCPs and (3) using these rectangular PCPs to strengthen (and significantly simplify) the rigid-matrix construction in Theorem 1.1. We begin by stating the improved rigid matrix construction.

Theorem 1.2. *There is a constant $\delta \in (0, 1)$ such that there is an FNP-machine that for infinitely many N, on input 1^N outputs an $N \times N$ matrix that is $(\delta \cdot N^2, 2^{\log N/\Omega(\log \log N)})$-rigid.*

We remark that Alman and Chen obtained a conditional result which proved a similar conclusion using the easy witness lemma of Impagliazzo, Kabanets and Wigderson [IKW02]: either $\mathsf{NQP} \not\subset \mathsf{P/poly}$ or for all $\varepsilon \in (0, 1)$ there exists an FNP-algorithm that for infinitely many N, on input 1^N outputs an $N \times N$ matrix that is $(\delta \cdot N^2, 2^{(\log N)^{1-\varepsilon}})$-rigid. Our main result (Theorem 1.2) is thus a common strengthening of both Theorem 1.1 as well as this conditional result.

1.2 Rectangular PCP

Our result is obtained by a new notion of PCPs, called *rectangular PCPs*.[5] Briefly put, rectangular PCPs are PCPs where the proofs are thought of as square matrices, and the random coins used by the verifier can be partitioned into two disjoint sets, one determining the *row* of each query and the other determining the *column*. To get a better feel for this new property, we examine the constraint satisfaction problem (CSP) underlying a rectangular PCP,[6] and defer the full definition to Section 2.

Consider the classical NP-hard constraint satisfaction problem MAXCUT whose instance is a directed graph $G = (V, E)$ with n vertices and m edges and the goal is to find a subset $S \subseteq V$ that maximizes the number of edges cut (in either direction) between S and $V \setminus S$. The instance G is *rectangular* if the following condition is met.

- There exist two directed graphs G_1 and G_2 with $\ell = \sqrt{n}$ vertices and $r = \sqrt{m}$ edges each such that G is the product graph $G_1 \times G_2$, i.e., the edges of G satisfy the following *rectangular* property:

$$((u_1, u_2), (v_1, v_2)) \in E(G) \iff (u_1, v_1) \in E(G_1) \text{ and } (u_2, v_2) \in E(G_2).$$

---

[5] We thank Ramprasad (RP) Saptharishi for suggesting the term "rectangular PCPs".

[6] See, for example, [AB09, Chapter 18] for a description of the CSP underlying a PCP.

3

An instance G is said to be τ-*almost rectangular* for $\tau \in [0, 1)$ if G is the edge-disjoint union of m^τ product graphs $G_1^{(j)} \times G_2^{(j)}, j \in [m^\tau]$ where each of the product graphs $G_1^{(j)} \times G_2^{(j)}$ is defined on the same vertex set V and satisfies $|E(G_1^{(j)})| = |E(G_2^{(j)})| = m^{(1-\tau)/2}$. To distinguish rectangular graphs from almost-rectangular graphs, we will sometimes refer to them as perfectly rectangular.

This definition of rectangularity can be extended to arbitrary q-CSPs as follows. Let Φ be a q-CSP instance on a set V of $n = \ell^2$ variables. Let $\mathcal{C}$ be the set of $m = r^2$ constraints of Φ. As both the number of variables ($n = \ell^2$) and the number of constraints ($m = r^2$) are perfect squares, we will w.l.o.g. index them with double indices, $(i_i, i_2) \in [\ell] \times [\ell]$ and $(j_1, j_2) \in [r] \times [r]$. Let the (j_1, j_2)-th constraint in $\mathcal{C}$ involve the q variables $x_{c_1(j_1, j_2)}, \ldots, x_{c_q(j_1, j_2)}$. The instance Φ is said to be rectangular if for any $k \in [q]$, the address function $c_k : [r] \times [r] \to [\ell] \times [\ell]$ that specifies the k-th variable in the (j_1, j_2)-th clause can be decomposed into a product function $a_k \times b_k$ where $a_k, b_k : [r] \to [\ell]$. Almost rectangularity is defined similarly.

Back in the "proof systems" view, a PCP is said to be *(τ-almost) rectangular* if its underlying CSP is (τ-almost) rectangular. Thus, rectangularity can be viewed as natural structural property referring to the clause-variable relationship in the CSPs produced by the PCP. Our main technical result is that there exists an efficient, short, smooth and almost-rectangular PCP. A simplified version of our result is as follows (see Theorem 8.2 for the exact statement.)

Theorem 1.3. *Let L be a language in $\mathsf{NTIME}(2^n)$. For every constants $s \in (0, 1/2)$ and $\tau \in (0, 1)$, there exists a constant-query, smooth and τ-almost rectangular PCP for L over the Boolean alphabet with soundness error s, proof length at most $2^n \cdot \mathsf{poly}(n)$ and verifier running time at most $2^{O(\tau n)}$.*

1.3 Rectangular PCPs to rigid matrices

We now sketch how rectangular PCPs can be used to construct rigid matrices. This will also serve as a motivation for the definition of rectangular PCPs. Our construction follows that of Alman and Chen, with the main difference being the use of rectangular PCPs. We show that this simplifies their construction and improves the rigidity parameters it attains. The construction is inspired by the subtitular maxim:

> *Hard claims have complex proofs.*

Informally speaking, the construction is an instantiation of this maxim where "complexity" refers to rigidity, "proofs" are PCPs, and "hard claims" are instances of the hard language guaranteed by the non-deterministic time hierarchy theorem (see next).

The main ingredients in our construction are as follows:

1. The non-deterministic time-hierarchy theorem [Zák83]: There exists a unary language $L \in \mathsf{NTIME}(2^n) \setminus \mathsf{NTIME}(2^n/n)$.

2. A non-trivial (i.e., sub-quadratic time) algorithm to compute the number of ones in a low-rank $\{0, 1\}$-valued matrix when given as input its low-rank decomposition $N = P \cdot Q$, where P and Q are matrices of dimensions $N \times \rho$ and $\rho \times N$, respectively. Such results with running time $N^{2-\varepsilon(\rho)}$ were developed by Chan and Williams [CW16] with $\varepsilon(\rho) = \Omega(1/\log \rho)$.

3. The existence of efficient, short, smooth and rectangular PCPs for $\mathsf{NTIME}(2^n)$, as guaranteed by Theorem 1.3.

Let $L \in \mathsf{NTIME}(2^n) \setminus \mathsf{NTIME}(2^n/n)$ as guaranteed by the non-deterministic time-hierarchy theorem. By Theorem 1.3, there exist efficient, short and smooth rectangular PCPs for L. Our goal is to show that either (a natural transformation of) the PCP yields a rigid matrix, or $L \in \mathsf{NTIME}(2^n/n)$ – a contradiction.

For simplicity of presentation, we will assume that the rectangular PCPs obtained in Theorem 1.3 are perfectly rectangular and furthermore that the underlying CSP of the PCP is MAXCUT with completeness c and soundness s for some constants $0 < s < c < 1$. In other words, the PCP reduction reduces instances $x \in L$ to digraphs G which have a fractional cut of size at least c, and instances $x \notin L$ to digraphs G which do not have any cut of fractional size larger than s.

Let us understand what it means for the PCP to be short, smooth, efficient and rectangular: *"Rectangular"* refers to the fact that the digraph G is a product graph $G_1 \times G_2$; *"Short"* implies that

the size of G (i.e., the number of vertices and edges) is at most $r^2 = 2^n \cdot \mathsf{poly}(n)$; *"Smooth"* implies that the digraph G is regular (the degree of a vertex is the sum of its in-degree and out-degree); and *"Efficient and rectangular"* implies that for each of the two graphs G_1 and G_2, given an edge the vertices incident on the edge can be obtained in time $2^{\gamma n}$ (for a small constant $\gamma > 0$ of our choice).

For any instance x of the language L, any cut of the corresponding graph G is of the form $(S, V(G) \setminus S)$. Since $V(G) = V(G_1) \times V(G_2)$, we can identify the cut S with a $V(G_1) \times V(G_2)$-matrix with $\{0,1\}$ entries. Let $L_1, R_1 \in \{0,1\}^{E(G_1) \times V(G_1)}$ be the incidence matrices indicating the left and right endpoints of the edges in G_1 (i.e., if $e = (u,v) \in E(G_1)$ then $L_1(e,u) = R_1(e,v) = 1$). Similarly, define matrices $L_2, R_2 \in \{0,1\}^{E(G_2) \times V(G_2)}$. These matrices will not be computed explicitly; efficiency of the PCP implies that for any given row-index, the non-zero column of that row can be computed in time $2^{\gamma n}$. We will refer to this fact as the *somewhat-efficient* computation of these matrices.

Observe that the matrix $L_1 \cdot S \cdot L_2^T$ is an indicator matrix indicating if the left endpoint of the edge is in the set S or not. Similarly $R_1 \cdot S \cdot R_2^T$ refers to the indicator of the right endpoint of the edge. Hence, the matrix $M(S) := L_1 \cdot S \cdot L_2^T + R_1 \cdot S \cdot R_2^T$ is the indicator matrix of whether the edge is cut by the set S or not (with addition over $\mathbb{F}_2$). Thus, the size of the cut induced by the set S is exactly the number of ones in the matrix $M(S)$. Let us denote this quantity by $\mathrm{val}(S) := \#_1(M(S))$.

We will prove that for infinitely many $x \in L$, every cut $S^* \in \{0,1\}^{V(G_1) \times V(G_2)}$ of fractional size at least c is a $(\delta \cdot N^2, \rho)$-rigid matrix, for $\delta = (c-s)/3$ and $\rho = 2^{n/\Omega(\log n)}$. Assume towards contradiction that this was not the case. Then, there for long enough $x \in L$, there exists a cut S^* that is non-rigid. Since S^* is non-rigid, it is δ-close to some Boolean matrix $S = P \cdot Q$ such that P and Q are Boolean matrices of dimensions $V(G_1) \times \rho$ and $\rho \times V(G_2)$, respectively. We now make two observations.

- Since the cut S^* is of size at least c and S is δ-close to S, it follows from the regularity of G that the cut induced by the set S is of size at least $c - 2\delta$. In other words, $\mathrm{val}(S) \geq c - 2\delta$.

- We can compute $\mathrm{val}(S) = \#_1(M(S))$ in time $O\left(r \cdot (\rho + 2^{\gamma n}) + r^{2-\varepsilon(2\rho)}\right)$ as follows. Recall that $M(S) = L_1 \cdot S \cdot L_2^T + R_1 \cdot S \cdot R_2^T$ and $S = P \cdot Q$. Hence,

$$M(S) = L_1 \cdot P \cdot Q \cdot L_2^T + R_1 \cdot P \cdot Q \cdot R_2^T = \underbrace{\begin{pmatrix} L_1 & R_1 \end{pmatrix} \cdot \begin{pmatrix} P & 0 \\ 0 & P \end{pmatrix}}_{=: \widetilde{P}} \cdot \underbrace{\begin{pmatrix} Q & 0 \\ 0 & Q \end{pmatrix} \cdot \begin{pmatrix} L_2^T \\ R_2^T \end{pmatrix}}_{=: \widetilde{Q}}$$

  So $M(S)$ is a matrix of rank at most 2ρ with a low-rank decomposition given by $M(S) = \widetilde{P} \cdot \widetilde{Q}$. Given matrices P and Q and the somewhat-efficient computation of the matrices L_1, L_2, R_1, R_2, the matrices $\widetilde{P}$ and $\widetilde{Q}$ may be computed in time $r \cdot (\rho + 2^{\gamma n})$. Finally, we invoke the algorithm of Chan and Williams to compute $\mathrm{val}(S) = \#_1(\widetilde{P} \cdot \widetilde{Q})$ in time $O\left(r^{2-\varepsilon(2\rho)}\right)$.

This suggests the following non-deterministic algorithm for checking membership in L

- On input 1^n

  1. Non-deterministically guess matrices $P \in \{0,1\}^{\ell \times \rho}$ and $Q \in \{0,1\}^{\rho \times \ell}$.
  2. Use the efficient PCP verifier to somewhat-efficiently compute the matrices L_1, L_2, R_1, R_2.
  3. Compute the matrices $\widetilde{P}$ and $\widetilde{Q}$.
  4. Compute the number of ones ν of the matrix $\widetilde{P} \cdot \widetilde{Q}$.
  5. Accept if and only if $\nu > s \cdot r^2$.

Indeed, this algorithm decides L: If $1^n \in L$, then there exists a guess $S = P \cdot Q$ that would get $\mathrm{val}(S) \geq (c - 2\delta) \cdot r^2 > s \cdot r^2$. On the other hand, if $1^n \notin L$, then by the soundness of the PCP, any cut S would have $\mathrm{val}(S) \leq s \cdot r^2$.

However, for a suitable choice of ρ, this algorithm runs in time $O\left(\ell \cdot \rho + r \cdot (\rho + 2^{\gamma n}) + r^{2-\varepsilon(2\rho)}\right) = O(2^n/n)$ – contradicting the time-hierarchy theorem. Hence, it is false that for every long enough $x \in L$, there exists a cut $S^* \in \{0,1\}^{V(G_1) \times V(G_2)}$ of fractional size at least c which is a non-rigid matrix. This immediately yields an FNP-algorithm that infinitely often outputs rigid matrices.

In Section 3 we complete this sketch into a full proof that deals with two significant caveats: the PCP is only almost-rectangular, and the predicate of the PCP is not necessarily MAXCUT. The first is not a significant obstacle and the generalization is rather immediate. The second requires more care, but examining the proof reveals that we only used the fact that each clause has the same predicate or, in PCP jargon, that the predicate is *oblivious to the randomness*. To this end, we define a property of PCPs wherein the predicates have randomness-oblivious-predicates (ROP) and show that the rectangular PCPs constructed in Theorem 1.3 can also be made ROP (see Sections 6 and 8 for exact details).

Comparison with the Alman–Chen construction: Alman and Chen obtained a similar result conditioned on the assumption $\mathsf{NQP} \not\subseteq \mathsf{P/poly}$, using the easy witness lemma. To obtain an unconditional result, they used a bootstrapping argument which results in rigidity for rank at most $2^{(\log N)^{1/4-\varepsilon}}$. The above proof, on the other hand, is not conditioned on any assumption, does not require the easy witness lemma, and implies rigid matrices for rank $2^{\log N/\Omega(\log \log N)}$. In fact, there is almost no loss due to the PCPs in the above argument. For instance, if the number of ones in $N \times N$ matrices of rank $N^{0.999}$ could be computed in sub-quadratic time, then our construction would yield matrices rigid for rank $N^{0.99}$.

1.4 Constructing rectangular PCPs

The rectangular property of PCPs states that the underlying constraint satisfaction problem (CSP) has a product structure.

Warm-up: Rectangularity of some known constructions

As a warm-up, let us examine the rectangularity of some common PCP building blocks.

First, it is immediate that PCPs obtained from *parallel repetition* are rectangular. Unfortunately, the size of these PCPs are far from being nearly-linear.

Next, recall the Blum–Luby–Rubinfeld (BLR) *linearity tester* [BLR93] that checks if a given function $f : \mathbb{F}_2^m \to \mathbb{F}_2$ is linear.

Algorithm (BLR Tester). On oracle access to $f \colon \mathbb{F}_2^m \to \mathbb{F}_2$,

1. Sample $x, y \in_R \mathbb{F}_2^m$.

2. Query f at locations x, y, and $x + y$.

3. Accept if and only if $f(x) + f(y) + f(x + y) = 0$.

The (x, y)-th constraint in the above test queries the three locations $x, y, x + y \in \mathbb{F}_2^{3m}$. For even m, we can write $x = (x_1, x_2)$ and $y = (y_1, y_2)$ where $x_1, x_2, y_1, y_2 \in \mathbb{F}_2^{m/2}$. Thus, the $((x_1, x_2), (y_1, y_2))$-test queries the three locations (x_1, x_2), (y_1, y_2) and $(x_1 + y_1, x_2 + y_2)$. Hence, the BLR test is perfectly rectangular. For similar reasons, the low-degree test is also perfectly rectangular.

The actual construct

The warm-up gives us hope that PCPs constructed using from low-degree test are rectangular or can be made so with some modification. Our construction, essentially, realizes this hope. In particular we take a closer look at the short and efficient PCP construction of Ben-Sasson *et al.* [BGHSV06, BGHSV05] and modify it suitably to obtain a rectangular PCP. This is a rather delicate operation and involves several subtleties along the way. We highlight the salient steps in the construction below.

For starters, recall another key ingredient in the construction of PCPs: the composition paradigm of Arora and Safra [AS98]. We will use the modular composition paradigm of Ben-Sasson *et al.* [BGHSV06] and Dinur and Reingold [DR06], wherein a *robust* PCP is composed with a PCP *of proximity*. Our construction of rectangular PCPs will proceed along the following lines.

1. We first show that the Reed–Muller based PCP construction due to Ben-Sasson *et al.* [BGHSV06, BGHSV05] can be modified to yield a short almost-rectangular robust PCP. This involves a careful,

step-by-step examination of this PCP. As indicated above, the low-degree component of this PCP is perfectly rectangular. However, this PCP also involves a sum-check component, which is inherently *not* rectangular, but is fortunately *almost* rectangular.

2. We then show that composition of an almost-rectangular robust PCP with a (not necessarily rectangular) PCP of proximity yields an almost-rectangular PCP.

Composing the outer robust PCP obtained in Item 1 with the short and efficient PCP of proximity of Mie [Mie09] yields a short, efficient and rectangular PCP with constant query complexity. However, this PCP is not necessarily smooth. By now, there are several standard techniques to "smoothify" a PCP in literature, but these techniques do not necessarily retain the rectangular property. To obtain a rectangular and smooth PCP, we actually work with a stronger notion of rectangularity, that we refer to as "rectangular-neighborhood-listing (RNL)" and show that a short and efficient PCP with RNL can be "smoothified" to yield the desired short, efficient, smooth and rectangular PCP.

Related Work: Independently and concurrently of our work, following the work of Alman and Chen [AC19], Viola [Vio20] showed the existence of functions on n variables in E^{NP} with approximate probabilistic degree $\Omega(n/\log^2 n)$ over $\mathbb{F}_2$, for infinitely many n. Using the known relation between matrix rigidity and approximate rank (see [AC19, Proposition 7.5]), our main theorem on matrix rigidity implies a similar lower bound on the approximate probabilistic degree.

Organization

The rest of the paper is organized as follows.

Preliminaries (Section 2). We begin by giving a definitional treatment of PCPs and its variants. In particular, we formally define the rectangular PCP, which is the central object of our focus. We also define the two aforementioned related properties: rectangular-neighborhood-listing (RNL) and randomness-oblivious-predicates (ROP).

From Rectangular PCPs to Rigid Matrices (Section 3). We show how the existence of efficient, short and smooth rectangular PCPs with ROP for $\mathsf{NTIME}(2^n)$ yields rigid matrices (thus proving Theorem 1.2, modulus the actual rectangular PCP construction).

A Construction of Rectangular PCPs (Sections 4 to 8). In the remaining sections of the paper, we construct efficient, short and smooth rectangular PCPs for $\mathsf{NTIME}(T(n))$. The main steps in the construction are as follows:

- Section 4: We show how any PCP with RNL and ROP can be converted to a smooth and rectangular PCP with ROP. Hence, from this point onwards, we seek PCPs with RNL, rather than rectangular PCPs.
- Section 5: We show that the robust PCP verifier of Ben-Sasson *et al.* [BGHSV06, BGHSV05] has RNL.
- Section 6: We show how to add ROP to any robust PCP with RNL.
- Section 7: We then show that any PCP of proximity, when composed with a robust PCP that has RNL and ROP, yields a PCP with RNL and ROP. Note that the PCP of proximity need not be rectangular.
- Section 8: Finally, we combine the results proved in Sections 4 to 7 to obtain our main construct: an efficient, short and smooth rectangular PCP (thus proving Theorem 1.3).

2   PCPs: definitions and variants

The main focus of this section is to introduce the notion of *rectangular PCPs*, the central object of interest in this work. To this end, we begin by recalling the standard definition of PCPs and related objects (PCP verifier, PCPs of proximity, robust soundness, smooth PCPs) before proceeding to define rectangular PCPs.

Notation. Let Σ be any finite alphabet. For $u, v \in \Sigma^n$, the relative Hamming distance between u and v, denoted by $\delta(u, v)$, is the fraction of locations on which u and v differ (i.e., $\delta(u, v) := |\{i : u_i \neq v_i\}|/n$). We say that u is δ-*close* to v (resp., δ-*far* from v) if $\delta(u, v) \leq \delta$ (resp., $\delta(u, v) > \delta$). The relative distance of a string u to a set V of strings is defined as $\delta(u, V) := \min_{v \in V}\{\delta(u, v)\}$.

2.1 Standard PCPs

We begin by recalling the formalism of an efficient PCP verifier. As is standard in this literature, we restrict our attention to *non-adaptive* verifiers.

Definition 2.1 (efficient PCP verifiers)**.**

- *Let* $r, q, m, d, t, \sigma \colon \mathbb{N} \to \mathbb{N}$. *A* (r, q, m, d, t)-restricted verifier *over alphabet* $\Sigma := \{0, 1\}^\sigma$ *is a probabilistic algorithm* V *that, on an input* x *of length* n, *tosses* $r := r(n)$ *random coins* R *and generates a sequence of* $q := q(n)$ query locations $I := (i^{(1)}, \ldots, i^{(q)})$, *where each* $i^{(k)} \in [m(n)]$, *and a* (decision) predicate $D \colon \Sigma^q \to \{0, 1\}$ *of size at most* $d(n)$ *in time at most* $t(n)$.

  Think of V *as representing a probabilistic oracle machine that queries the proof oracle* $\pi \in \Sigma^m$, *for the positions in* I, *receives the* q *symbols* $\pi|_I := (\pi_{i^{(1)}}, \ldots, \pi_{i^{(q)}})$, *and accepts iff* $D(\pi|_I) = 1$.

- *We write* $(I, D) \xleftarrow{R} V(x)$ *to denote the queries and predicate generated by* V *on input* x *and random coin tosses. To explicitly mention the random coins* R, *we write* $(I, D) \leftarrow V(x; R)$.

- *We call* r *the* randomness complexity, q *the* query complexity, m *the* proof length, d *the* decision complexity *and* t *the* running time *of* V. σ *is called the* answer complexity *of* V, *and will usually be omitted.*[7]

It will be convenient at times to have the following graphical description of the verifier. Given a (r, q, m, d, t)-restricted verifier and input x, consider the bipartite graph $G(V, x) := (L = \{0, 1\}^r, R = [m], E)$ where $(R, i) \in E$ if the verifier V on input x and random coins R queries location i in the proof. Clearly, the graph $G(V, x)$ is q-left regular.

We can now define the standard notion of efficient PCPs with perfect completeness.

Definition 2.2 (PCP)**.** *For a function* $s : \mathbb{N} \to [0, 1]$, *a verifier* V *is a* probabilistically checkable proof system (PCP) *for a language* L *with* soundness error s *if the following two conditions hold for every string* x:

Completeness: *If* $x \in L$ *then there exists* π *such that* $V(x)$ *accepts oracle* π *with probability 1. Formally,*

$$\exists \pi \quad \Pr_{(I,D) \xleftarrow{R} V(x)} [D(\pi|_I) = 1] = 1.$$

Soundness: *If* $x \notin L$ *then for every oracle* π, *the verifier* $V(x)$ *accepts* π *with probability strictly less than* s. *Formally,*

$$\forall \pi \quad \Pr_{(I,D) \xleftarrow{R} V(x)} [D(\pi|_I) = 1] < s(|x|).$$

While constructing PCPs, we will sometimes be interested in PCPs with a stronger notion of soundness, referred to as *robust soundness*.

Definition 2.3 (robust soundness)**.** *For functions* $s, \rho : \mathbb{N} \to [0, 1]$, *a PCP verifier* V *for a language* L *has* robust-soundness error s *with* robustness parameter ρ *if the following holds for every* $x \notin L$: *For every oracle* π, *the symbols read by the verifier* V *are* ρ-*close to being accepted with probability strictly less than* s. *Formally,*

$$\forall \pi \quad \Pr_{(I,D) \xleftarrow{R} V(x)} [\exists a \ s.t. \ D(a) = 1 \ and \ \delta(a, \pi|_I) \leq \rho] < s(|x|).$$

By now, we know of several such efficient PCP constructions, one of which we state below.

---

[7]All PCPs in this work will be Boolean (i.e., $\sigma = 1$), except for an intermediate PCP in Section 5. Even there, it will be more convenient to consider the alphabet size $|\Sigma| = 2^\sigma$ rather than σ.

Theorem 2.4 (efficient PCPs for $\mathsf{NTIME}(T)$ [BGHSV05, Theorem 2.6]). *Suppose that L is a language in $\mathsf{NTIME}(T(n))$ for some non-decreasing function $T \colon \mathbb{N} \to \mathbb{N}$. Then for every $\varepsilon \in (0,1)$, L has a PCP verifier over $\{0,1\}$ with soundness error ε, query complexity $O(1/\varepsilon)$ and randomness complexity $\log T(n) + \log^{O(\varepsilon)} T(n)$.*

While constructing variants of the above PCP, we will particularly be interested in smooth PCPs.

Definition 2.5 (smooth PCP). *Given a (r, q, m, d, t)-restricted verifier V, an input x and $i \in [m]$, let $Q_x(i)$ denote the probability with which the verifier V outputs i on a random query $k \in [q]$. Formally,*

$$Q_x(i) := \Pr_{R, k \in [q]}[i^{(k)} = i | (I, D) \leftarrow V(x; R)].$$

The PCP verifier V is said to be smooth if for all $i, j \in [m]$, $Q_x(i) = Q_x(j)$.

Thus, smooth PCPs refer to PCPs whose verifiers query all locations of the proof oracle equally likely (or equivalently in the above graphical description, verifiers whose corresponding bipartite graphs are also right-regular).[8]

Remark 2.6 (tolerance of smooth PCPs). *A smooth PCP is tolerant of errors in a correct proof, in the sense that a proof that is close to a correct one is accepted with good probability. Concretely, suppose V makes q queries to its proof and is smooth. Then if π is a correct proof for V (i.e. accepted w.p. 1) and π^* is δ-close to π in relative Hamming distance, then π^* is accepted with probability at least $1 - q \cdot \delta$.*

The PCPs constructed in Theorem 2.4 are not necessarily smooth, however they can be made smooth without too much of an overhead. In this work we will be interested in *smoothening* the PCP maintaining yet another property, *rectangularity*, which we introduce in the following section.

2.2 Rectangular PCPs

We now define *rectangular PCPs*, the central object of interest in this work. As the name suggests, rectangular PCPs are PCPs in which the proof oracle $\pi : [m] \to \Sigma$, an m-length string, is interpreted as a matrix $\pi : [\ell] \times [\ell] \to \Sigma$ for some ℓ such that $m = \ell^2$ (yes, we assume that the proof lengths are always squares of integers). Furthermore, the verifier is also "rectangular" in the sense that the randomness $R \in \{0,1\}^r$ is also partitioned into 2 parts $R = (R_{\mathrm{row}}, R_{\mathrm{col}})$ such that the row index of the queries is obtained from the "row randomness" R_{row} while the column index of the queries is obtained from the "column randomness" R_{col}.

The above informal description assumes "perfect" rectangularity while the definition below allows for the relaxed notion of "almost-rectangularity", in which randomness is partitioned into three parts: row and column (as above), as well as a small *shared* part that is used for obtaining both the rows and the columns of the queries.

Definition 2.7 (Rectangular PCP). *For $\tau \in [0,1)$, a (r, q, ℓ^2, d, t)-restricted verifier V is said to be τ-rectangular if the following holds.*

The random coin tosses $R \in \{0,1\}^r$ can be partitioned into 3 parts

$$R = (R_{\mathrm{row}}, R_{\mathrm{col}}, R_{\mathrm{shared}}) \in \{0,1\}^{(1-\tau)r/2} \times \{0,1\}^{(1-\tau)r/2} \times \{0,1\}^{\tau r},$$

such that the verifier V on input x of length n and random coins R produces a sequence of q proof locations $I = ((i_{\mathrm{row}}^{(1)}, i_{\mathrm{col}}^{(1)}), \ldots, (i_{\mathrm{row}}^{(q)}, i_{\mathrm{col}}^{(q)}))$ as follows:

- $I_{\mathrm{row}} := (i_{\mathrm{row}}^{(1)}, \ldots, i_{\mathrm{row}}^{(q)}) = V_{\mathrm{row}}(x; R_{\mathrm{row}}, R_{\mathrm{shared}})$,

- $I_{\mathrm{col}} := (i_{\mathrm{col}}^{(1)}, \ldots, i_{\mathrm{col}}^{(q)}) = V_{\mathrm{col}}(x; R_{\mathrm{col}}, R_{\mathrm{shared}})$,

---

[8] *Minor historical inconsistencies in the definition of smoothness:* Several previous works [KT00, Par20] defined a smooth oracle machine as one in which each location of the oracle has equal probability of being queried by the machine *in any of its queries* (rather than in a random query, as in Definition 2.5 as well as other prior works [GS06, BGHSV06]). Indeed, both definitions are equivalent assuming the machine never queries the same location twice for any given random coin sequence R. Our definition is more convenient as it coincides with right-regularity of the corresponding bipartite graph even without this assumption.

- *Generating I_{row}, I_{col}, and the decision predicate[9] take a total of at most $t(n)$ time.*

In other words, the row (respectively column) indices of the queries are only a function of the row (respectively column) and shared parts of the randomness. If $\tau = 0$, we will say the verifier V is **perfectly rectangular**, and otherwise V is **almost rectangular**. When it is obvious from context, we will say that V is simply **rectangular**, omitting the "τ-" qualifier.

2.3 Rectangular Neighbor-Listing (RNL)

A careful reading of the construction of PCPs mentioned in Theorem 2.4 will reveal that they are in fact rectangular. However, for our application, we will need rectangular PCPs that are also smooth. Later, we will "smoothen" a PCP while maintaining its rectangularity (see Section 4), for which we need a stronger property that we refer to as *rectangular-neighbor-listing (RNL)*. To define this property, we first define *configurations* and *neighboring configurations*.

Definition 2.8 (configurations and neighboring configurations)**.** *Given a (r, q, m, d, t)-restricted verifier V and an input x, a configuration refers to a tuple $(R, k) \in \{0,1\}^r \times [q]$ composed of the randomness of the verifier and query index. The verifier V describes how to obtain the query location $i^{(k)} \in [m]$ from the configuration (R, k) (and the input x).*

*We say that two configurations (R, k) and (R', k') of a PCP verifier V on input x are **neighbors** if they both yield the same query location $i \in [m]$. (In particular, every configuration is a neighbor of itself.)*

In the graphical representation of a verifier, a configuration refers to an edge of the bipartite graph and two configurations are said to be neighbors if they are incident on the same right vertex. A configuration $(R, k) = (R_{\text{row}}, R_{\text{col}}, R_{\text{shared}}, k)$ of a rectangular PCP can be broken down into a **row configuration** $(R_{\text{row}}, R_{\text{shared}}, k)$ and a **column configuration** $(R_{\text{col}}, R_{\text{shared}}, k)$. Rectangularity states that the query location $(i_{\text{row}}^{(k)}, i_{\text{col}}^{(k)}) \in [\ell] \times [\ell]$ satisfy that $i_{\text{row}}^{(k)}$ is a function of the row configuration while $i_{\text{col}}^{(k)}$ is a function of the column configuration.

Definition 2.9 (rectangular neighbor-listing (RNL))**.** *For $\tau \in [0, 1)$ and $t_{\text{RNL}} : \mathbb{N} \to \mathbb{N}$, an (r, q, m, d, t)-restricted verifier V is said to have the τ-**rectangular neighbor listing property** (τ-RNL) with time $t_{\text{RNL}}(n)$ if the following holds.*

- *The random coin tosses $R \in \{0,1\}^r$ can be partitioned into 4 parts*

$$R = (R_{\text{row}}, R_{\text{col}}, R_{\text{shared.row}}, R_{\text{shared.col}}) \in \{0,1\}^{(1-\tau)r/2} \times \{0,1\}^{(1-\tau)r/2} \times \{0,1\}^{\tau r/2} \times \{0,1\}^{\tau r/2},$$

  *where we refer to the 4 parts $R_{\text{row}}, R_{\text{col}}, R_{\text{shared.row}}, R_{\text{shared.col}}$ as the **row part**, **column part**, **row-shared part** and **column-shared part** respectively. We will refer to the combined shared randomness $R_{\text{shared}} := (R_{\text{shared.row}}, R_{\text{shared.col}})$ as the **shared part**.*

- *There exist two algorithms, a **row agent** (denoted A_{row}) and a **column agent** (denoted A_{col}) that list, in time $t_{\text{RNL}}(n)$, all neighbors of a given configuration (R, k) in the following "rectangular and synchronized" fashion:*

  - *On input a row configuration $(R_{\text{row}}, R_{\text{shared}}, k)$, the row agent A_{row} outputs a list L_{row} of tuples $(R'_{\text{row}}, R'_{\text{shared.row}}, k')$.*
  - *On input a column configuration $(R_{\text{col}}, R_{\text{shared}}, k)$, the column agent A_{col} outputs a list L_{col} of tuples $(R'_{\text{col}}, R'_{\text{shared.col}}, k')$.*

  satisfying the following properties

  1. *The two lists L_{row} and L_{col} are of equal length and entrywise-matching k' values, such that the "zipped" list*

$$L := \left\{ (R'_{\text{row}}, R'_{\text{col}}, R'_{\text{shared.row}}, R'_{\text{shared.col}}, k') \;\middle|\; \begin{array}{c} i \in [|L_{\text{row}}|] \\ (R'_{\text{row}}, R'_{\text{shared.row}}, k') := L_{\text{row}}[i] \\ (R'_{\text{col}}, R'_{\text{shared.col}}, k') := L_{\text{col}}[i] \end{array} \right\} \tag{1}$$

---

[9]It is natural to wonder how the decision predicate depends on the randomness. This is considered in Section 2.4.

is the list of all full configurations that are neighbors of (R, k).

2. *Not only are the contents of L the same for each two neighboring locations, but the* order *of configurations in L is the same too. That is, for any two neighboring configurations (R, k) and $\left(\widetilde{R}, \widetilde{k}\right)$, the resulting configuration lists L and $\widetilde{L}$ are equal as ordered lists (element-by-element).*

3. *Both agents know the index of (R, k) in the list L (despite not knowing (R, k) entirely).*

Informally speaking, rectangularity asserts that the query location can be obtained in a "rectangular" fashion from the randomness, while RNL asserts that the entire list of neighboring configurations of the query location can be obtained in a "rectangular" fashion.

A PCP with RNL can be made *smooth* and *rectangular*, as shown in Section 4.

2.4 Randomness-oblivious predicates (ROP)

For our application of rectangular PCPs (Section 3), we would like the decision circuit to depend only on the shared part of the randomness. However, we do not know how to obtain such a PCP (that is also *smooth* and *short*), so we allow the decision circuit to take a limited number of *parity checks* of the entire randomness. Like the decision circuit, the choice of parity checks depends only on the shared part of the randomness.

Definition 2.10 (efficient PCP verifiers with τ-ROP)**.** *For $\tau \in [0, 1)$, a (r, q, m, d, t)-restricted verifier V is said to have the τ-randomness-oblivious predicates (τ-ROP) if the following holds.*

The random coin tosses $R \in \{0, 1\}^r$ can be partitioned into two parts

$$R = (R_{\mathrm{obliv}}, R_{\mathrm{aware}}) \in \{0, 1\}^{(1-\tau)r} \times \{0, 1\}^{\tau r},$$

such that the verifier V on input x of length n and random coins R runs in time $t(n)$, and

1. *Based only on R_{aware}:*

   (a) *Constructs a (decision) predicate $D \leftarrow V(x; R_{\mathrm{aware}})$ of size at most $d(n)$.*

   (b) *Constructs a sequence of randomness parity checks $(C_1, \ldots, C_p) \leftarrow V(x; R_{\mathrm{aware}})$, each of which is an affine function on $\{0, 1\}^{(1-\tau)r} \to \{0, 1\}$.[10]*

2. *Based on all of the randomness $R = (R_{\mathrm{obliv}}, R_{\mathrm{aware}})$, produces a sequence of q proof locations $I = (i^{(1)}, \ldots, i^{(q)})$, where each $i^{(k)} \in [m(n)]$.*

Think of V as representing a probabilistic oracle machine that queries proof oracle π and gets answer symbols $\pi|_I$, computes parity checks $P \coloneqq (C_1(R_{\mathrm{obliv}}), \ldots, C_p(R_{\mathrm{obliv}}))$ and accepts iff $D(\pi|_I, P) = 1$.

We write $(I, P, D) \xleftarrow{R} V(x)$ to denote the queries, predicate, and parities generated by V on input x. To explicitly mention the random coins R, we write $(I, P, D) \leftarrow V(x; R)$.

We call p the parity-check complexity of V.

We view ROP as a secondary property to RNL and rectangularity, and for simplicity we sometimes omit it from informal discussions (e.g., the title Section 3). Indeed, in Section 6 we show a simple way of adding adding ROP to any PCP while essentially increasing only its decision complexity.

2.5 A description of a rectangular verifier with ROP

All new PCP notions that are key to our work deal with a modified view of the run of a PCP verifier based on a partitioning of its randomness. Thus, let us take a moment to provide a streamlined description of a rectangular PCP verifier that has ROP. We hope this description helps the reader picture the new properties of our main PCP verifier, which we eventually construct in Theorem 8.2, and use in our construction of rigid matrices (Section 3). Specifically, we wish to clarify the dependence of the queries, the decision predicate and the parity checks on the different parts of the randomness.

---

[10]These are affine functions of the oblivious part only, but they encompass parities on *all* of the randomness by including the parity of the aware part in the constant term.

Remark 2.11 (Rectangular verifier with ROP). *Let $\tau \in (0,1)$, and let V be a τ-rectangular $\left(r, q, p, \ell^2, d, t\right)$- verifier with τ-ROP. Assume further that the* shared *and* aware *parts of the randomness of V are the same,[11] such that its randomness R is partitioned as follows:*

$$R_{\mathrm{obliv}} = (R_{\mathrm{row}}, R_{\mathrm{col}})$$
$$R_{\mathrm{aware}} = R_{\mathrm{shared}}$$
$$R = (R_{\mathrm{row}}, R_{\mathrm{col}}, R_{\mathrm{shared}}) = (R_{\mathrm{obliv}}, R_{\mathrm{aware}}).$$

Since the shared *and* aware *parts of the randomness are the same, we will refer only to the* shared *part of the randomness.*

The run of V given input x and proof oracle π can be described as follows:

1. *Sample* shared *randomness $R_{\mathrm{shared}} \in \{0,1\}^{\tau \cdot r}$. Based on it,*

   (a) *Construct a decision predicate $D := D(x; R_{\mathrm{shared}})$ of size d.*

   (b) *Construct randomness parity checks $(C_1, \ldots, C_p) := (C_1(x; R_{\mathrm{shared}}), \ldots, C_p(x; R_{\mathrm{shared}}))$.*

2. *Sample* row *randomness $R_{\mathrm{row}} \in \{0,1\}^{(1-\tau)r/2}$. Construct proof row locations*

$$i_{\mathrm{row}}^{(1)} := i_{\mathrm{row}}^{(1)}(x; R_{\mathrm{row}}, R_{\mathrm{shared}}), \ldots, i_{\mathrm{row}}^{(q)} := i_{\mathrm{row}}^{(q)}(x; R_{\mathrm{row}}, R_{\mathrm{shared}}).$$

3. *Sample* column *randomness $R_{\mathrm{col}} \in \{0,1\}^{(1-\tau)r/2}$. Construct proof column locations*

$$i_{\mathrm{col}}^{(1)} := i_{\mathrm{col}}^{(1)}(x; R_{\mathrm{col}}, R_{\mathrm{shared}}), \ldots, i_{\mathrm{col}}^{(q)} := i_{\mathrm{col}}^{(q)}(x; R_{\mathrm{col}}, R_{\mathrm{shared}}).$$

4. *Compute randomness parity checks $P := (C_1(R_{\mathrm{row}}, R_{\mathrm{col}}), \ldots, C_p(R_{\mathrm{row}}, R_{\mathrm{col}}))$. Query the proof oracle to obtain $\pi|_I := \left(\pi_{i_{\mathrm{row}}^{(1)}, i_{\mathrm{col}}^{(1)}}, \ldots, \pi_{i_{\mathrm{row}}^{(q)}, i_{\mathrm{col}}^{(q)}} \right)$.*

5. *Output the result of the computation $D(\pi|_I, P)$.*

3 From rectangular PCPs to rigid matrices

Alman and Chen [AC19] show how to construct rigid matrices using efficient, short and smooth PCPs. In this section, we show how efficient, short, smooth and *rectangular* PCPs can be used to obtain a simpler and stronger construction of rigid matrices.

Lemma 3.1. *Let $\tau \in (0,1)$ and $\rho : \mathbb{N} \to \mathbb{N}$. Let $L \in \mathsf{NTIME}(2^n) \setminus \mathsf{NTIME}(O(2^n/n))$ be a unary language. Suppose L has a PCP with soundness error s and a $\left(r, q, \ell^2, d, t\right)$-restricted verifier V over alphabet $\{0,1\}$ with $\ell(n) = \mathsf{poly}(2^n)$ strictly monotone increasing. Assume further that V is τ-rectangular and has τ-ROP with parity-check complexity p, and that the* shared *and the* aware *parts of the randomness are the same. Lastly, assume that the following inequalities hold:*

1. $\frac{1+\tau}{2} \cdot r + \log(t + \rho) \leq n - \log n$.

2. $q + p + r - \Omega\left(\frac{(1-\tau)r}{\log((q+p)\rho)}\right) \leq n - \log n$.

Then, there is an FNP*-machine such that, for infinitely many $N \in \mathbb{N}$, on input 1^N, outputs an $N \times N$ matrix with $\mathbb{F}_2$ entries which is $\left(\frac{1-s}{q} \cdot N^2, \rho(\ell^{-1}(N))\right)$-rigid.*

To prove Lemma 3.1, we make use of the following fast algorithm that counts the number of 1's in a low rank matrix (given its low rank decomposition).

Theorem 3.2 ([CW16, AC19]). *Given two matrices $A \in \mathbb{F}_2^{n \times \rho}$ and $B \in \mathbb{F}_2^{\rho \times n}$ where $\rho = n^{o(1)}$, there is a (deterministic) algorithm that computes the number of 1's in the matrix $A \cdot B$ in time $T(n, \rho) := n^{2 - \Omega\left(\frac{1}{\log \rho}\right)}$.*

---

[11] This will be the case in the rest of this work.

We now prove Lemma 3.1.

Proof of Lemma 3.1. Let V be the postulated PCP verifier for L. The FNP-machine computing rigid matrices (infinitely often) runs as follows. On input 1^N,

1. If $N = \ell(n)$ for some $n \in \mathbb{N}$: [12]

   (a) Guess an $N \times N$ matrix denoted by π.

   (b) Emulate $V^\pi(1^n)$ on all possible 2^r random coins. If V accepted on all randomness, accept and output π; else, reject.

2. Else ($N \neq \ell(n)$ for any n), reject.

This machine runs in time $O(2^r \cdot t) = \mathsf{poly}(2^n) = \mathsf{poly}(\ell(n)) = \mathsf{poly}(N)$, and whenever $N = \ell(n)$ for some n such that $1^n \in L$, one of its non-deterministic guesses lead to acceptance by completeness of the PCP. Note that there could be multiple non-deterministic guesses that lead to acceptance. We show that for infinitely many $N = \ell(n)$ such that $1^n \in L$, any guessed π that leads to acceptance is $\left(\frac{1-s}{q} \cdot N^2, \rho\right)$-rigid for $\rho := \rho(n) = \rho\big(\ell^{-1}(N)\big)$.

Assume towards contradiction that this is not the case. Then, there exists an n_0 such that for any $n \geq n_0$, $1^n \in L$ if and only if there exists a proof π (for the verifier V) which is $\frac{1-s}{q}$-close to a rank ρ matrix. We describe a non-deterministic algorithm that decides L in time $O(2^n/n)$ – a contradiction. Given input 1^n,

1. Guess matrices A and B of dimensions $\ell \times \rho$ and $\rho \times \ell$ respectively. (The right guess is when $A \cdot B$ is δ-close to π; by smoothness, the acceptance probability of $A \cdot B$ will then be close to that of π, and the task is reduced to estimating the acceptance probability of $A \cdot B$.)

2. Compute the acceptance probability of $A \cdot B$ by V as follows. For each sequence of coins in the shared part of the randomness $R_{\mathrm{shared}} \in \{0,1\}^{\tau \cdot r}$:

   (a) Compute the predicate $D := D(R_{\mathrm{shared}})$ and randomness parity checks $C_j := C_j(R_{\mathrm{shared}})$, $j \in [p]$.

   (b) **Prepare queries into proof:** For each $k \in [q]$,

      i. **Prepare left matrices:** Compute the $2^{(1-\tau)r/2} \times \rho$ matrix $A^{(k)}$ whose R_{row}-th row is just the row indexed by $i_{\mathrm{row}}^{(k)}(R_{\mathrm{row}}, R_{\mathrm{shared}})$ in A, for any $R_{\mathrm{row}} \in \{0,1\}^{(1-\tau)r/2}$.

      ii. **Prepare right matrices:** Compute the $\rho \times 2^{(1-\tau)r/2}$ matrix $B^{(k)}$ whose R_{col}-th column is just the column indexed by $i_{\mathrm{col}}^{(k)}(R_{\mathrm{col}}, R_{\mathrm{shared}})$ in B, for any $R_{\mathrm{col}} \in \{0,1\}^{(1-\tau)r/2}$.

      Observe that $\big(A^{(k)} \cdot B^{(k)}\big)_{R_{\mathrm{row}}, R_{\mathrm{col}}}$ is exactly the k-th bit read by the verifier on randomness $(R_{\mathrm{row}}, R_{\mathrm{col}}, R_{\mathrm{shared}})$.

   (c) **Prepare randomness parity checks:** For each $j \in [p]$,

      i. Compute the $2^{(1-\tau)r/2} \times 3$ matrix $A^{(q+j)}$ and the $3 \times 2^{(1-\tau)r/2}$ matrix $B^{(q+j)}$, for which $\big(A^{(q+j)} \cdot B^{(q+j)}\big)_{R_{\mathrm{row}}, R_{\mathrm{col}}} = C_j(R_{\mathrm{row}}, R_{\mathrm{col}})$. Such matrices exist and can be computed in time $O(r \cdot 2^{(1-\tau)r/2})$, as described in Claim B.1.

   (d) **Fast counting:** The task is to compute the acceptance probability of the (fixed) predicate D on inputs $\big(A^{(1)} \cdot B^{(1)}\big)_{R_{\mathrm{row}}, R_{\mathrm{col}}}, \ldots, \big(A^{(q+p)} \cdot B^{(q+p)}\big)_{R_{\mathrm{row}}, R_{\mathrm{col}}}$ for uniformly random R_{row} and R_{col}. Using Fourier analysis, it suffices to compute the acceptance probability of all parity predicates, i.e., to compute the number of 1's in $\bigoplus_{k \in K} A^{(k)} \cdot B^{(k)}$ for each $K \subseteq [q+p]$. For each K, note that $\bigoplus_{k \in K} A^{(k)} \cdot B^{(k)}$ is the product of a $2^{(1-\tau)r/2} \times (|K|\rho)$ and a $(|K|\rho) \times 2^{(1-\tau)r/2}$ matrix over $\mathbb{F}_2$ (namely, concatenate the rows of the $|K|$ matrices $\{A_k\}_{k \in K}$ and concatenate the columns of the $|K|$ matrices $\{B_k\}_{k \in K}$). Thus, for each K, computing the acceptance probability of $\bigoplus_{k \in K} A^{(k)} \cdot B^{(k)}$ can be done with the fast counting algorithm for low-rank matrices of Theorem 3.2. Its runtime is

$$T(2^{(1-\tau)r/2}, (q+p) \cdot \rho) = \left(2^{(1-\tau)r/2}\right)^{(2-\Omega(1/\log((q+p)\cdot\rho)))} = 2^{(1-\tau)r - \Omega((1-\tau)r/\log((q+p)\cdot\rho))}$$

---

[12]This can be done in time $O(n) = O(\log N)$ by finding the first integer n such that $\ell(n) \geq N$, and verifying that $\ell(n) = N$.

3. We have thus computed the acceptance probability of $A \cdot B$ by the verifier V. If this probability is at least the soundness error s, decide that the input 1^n is in L. Otherwise, decide that the input is not in L.

We claim that the algorithm correctly decides L, for input length $n \geq n_0$. Indeed, if $1^n \in L$ then, when guessing A and B such that $A \cdot B$ is $(\frac{1-s}{q})$-close to the correct proof π for 1^n (such A and B exist by our assumption towards contradiction), smoothness of the verifier V implies that its acceptance probability is at least $1 - q \cdot (\frac{1-s}{q}) = s$. On the other hand, if $1^n \notin L$, then soundness of V implies any guessed A and B leads to rejection with probability strictly less than s. Since n_0 is constant we can hard-wire the values of L on 1^n for $n < n_0$ so that the algorithm correctly decides L on all inputs.

As for its runtime, observe that Item 2a takes time $O(t)$, Item 2b takes time $O(2^{(1-\tau)r/2} \cdot (t + \rho))$ and Item 2c takes time $O(r \cdot 2^{(1-\tau)r/2})$. Since $t \geq r$, these are dominated by $O(2^{(1-\tau)r/2} \cdot (t + \rho))$. Therefore the runtime of the algorithm is

$$O\Big(2^{\tau \cdot r} \cdot \Big(2^{(1-\tau)r/2} \cdot (t + \rho) \ + \ 2^{q+p} \cdot 2^{(1-\tau)r - \Omega((1-\tau)r/\log((q+p)\cdot\rho))}\Big)\Big)$$
$$= O\Big(2^{(1+\tau)r/2} \cdot (t + \rho) \ + \ 2^{q+p+r - \Omega((1-\tau)r/\log((q+p)\cdot\rho))}\Big).$$

By the assumption on the parameters of the PCP, this is at most $O(2^n/n)$ – a contradiction. $\qquad\square$

Now that we formalized a connection between rectangular PCPs and rigid matrices, let us introduce the PCP that we construct in the remainder of this work, and show how it implies the rigid matrix construction asserted in Theorem 1.2.

Theorem 3.3 (Theorem 8.2, instantiated)**.** *For any $L \in \mathsf{NTIME}(2^n)$, and constants $s \in (0, 1/2)$ and $\tau \in (0,1)$, L has a PCP verifier over alphabet $\{0,1\}$ with the following parameters:*

- *Randomness complexity $r(n) = n + O(\log n)$*

- *Proof length $\ell(n)^2 = 2^n \cdot \mathsf{poly}(n)$.*

- *Soundness error s.*

- *Decision, query and parity-check complexities all $O(1)$.*

- *Verifier runtime $t(n) = 2^{O(\tau n)}$.*

- *The verifier is τ-rectangular and has τ-ROP. Furthermore, the shared and the aware parts of the randomness is the same.*

Theorem 3.3 is obtained by instantiating Theorem 8.2 with parameters $T(n) := 2^n$ and $m := \Omega(1/\tau)$. Next, we restate the rigid matrix construction asserted in Theorem 1.2 and reckon that it is obtained by combining Theorem 3.3 and Lemma 3.1.

Corollary 3.4 (Theorem 1.2, restated)**.** *There is a constant $\delta \in (0,1)$ such that there is an FNP-machine that for infinitely many N, on input 1^N outputs an $N \times N$ matrix that is $(\delta \cdot N^2, 2^{\log N/\Omega(\log\log N)})$-rigid.*

Proof. From [Zák83], there exists a unary language $L \in \mathsf{NTIME}(2^n) \setminus \mathsf{NTIME}(O(2^n/n))$. Let L be any such language. Fix $\rho(n) := 2^{n/(K\log n)}$ for a large enough constant K to be determined later. We verify that the parameters of the PCP of Theorem 3.3, for a sufficiently small $\tau \in (0,1)$, satisfy all the conditions from Lemma 3.1 for this ρ. Let q and p denote the query complexity and parity-check complexity of the PCP respectively. Set $\delta = (1-s)/q$. Now, for small enough τ,

$$\left(\tfrac{1+\tau}{2}\right) \cdot r + \log(\rho + t) \leq (1/2 + O(\tau))n + O(n/\log n) < n - \log n,$$

as required in Item 1. Also, the parameters satisfy Item 2, because

$$q + p + r - \Omega\left(\frac{(1-\tau)r}{\log((q+p)\rho)}\right) \leq n + O(\log n) - \Omega(K\log n) < n - \log n,$$

where the last inequality holds for a suitable choice of the constant K. As the proof length is $\ell(n)^2 = 2^n \cdot \mathsf{poly}(\log n)$, we have $\ell^{-1}(N) = \Theta(\log N)$. Therefore, $\rho(\ell^{-1}(N)) = 2^{\log N/\Omega(\log\log N)}$ and the corollary follows from Lemma 3.1. $\qquad\square$

Remark 3.5. *The only bottleneck preventing [Lemma 3.1](#) from giving rigid matrices for polynomial ranks, $\rho = N^{\Omega(1)}$, via [Theorem 3.3](#) is the runtime of the counting algorithm used in [Item 2d](#). That is, such results would be obtained if there was an algorithm for counting the number of nonzero entries in a rank $N^{\Omega(1)}$ matrix (of dimensions $N \times N$) that ran in time slightly better than $O(N^2)$. Our reduction would go through even if the algorithm's answer was only* approximately *correct (while losing the respective approximation factor in the distance of the resulting matrices from low rank ones). In fact, this seems to be the only bottleneck even up to rank $\rho = N^{1-O(\tau)}$.*

4 From RNL to smooth and rectangular PCPs

In this section, we show how any PCP verifier with RNL can be made into a smooth and rectangular PCP. This conversion preserves the ROP.

Theorem 4.1. *Suppose L has a PCP with verifier V_{old} as described in [Table 1](#), and τ-RNL and τ-ROP such that the* shared *and* aware *parts of the randomness are the same. Then for any $\mu \in (0,1)$, L has a PCP with verifier V_{new} as described in [Table 1](#) which is* smooth, *τ-rectangular and τ-ROP such that the* shared *and* aware *parts of the randomness are the same.*

Complexity	V_{old}	V_{new}
Alphabet size	2	2
Soundness error	s	$s + \mu$
Randomness	r	r
Query	q	$\mathsf{poly}(q/\mu)$
Parity-check	p	p
Proof length	m	$2^r \cdot q$
Decision	d	$d + \mathsf{poly}(q/\mu)$
Runtime	t	$t + q \cdot \mathsf{poly}(t_{\text{RNL}})$

Table 1: The complexities of the original V_{old} and the smooth verifier V_{new}. t_{RNL} is the running time of the row and column neighbor-listing agents of V_{old}.

The smooth proof system of [Theorem 4.1](#) utilizes an explicit construction of *sampler graphs*, defined and stated next.

Definition 4.2 (Sampler graph)**.** *Fix $\alpha \in [0,1]$. Graph $G = (V, E)$ is an α-sampler if for every $S \subseteq V$,*

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

Fact 4.3 ([[Gol11](#), Section 5.1])**.** *There exists an algorithm that given an integer n and $\alpha \in (0,1)$, constructs a $(4/\alpha^4)$-regular graph on n vertices which is an α-sampler in time $\mathsf{poly}(n)$.*

4.1 The smooth and rectangular PCP

The smooth and rectangular PCP verifier is obtained by applying the degree reduction transformation of [[DH13](#), Theorem 5.1]. We restate this transformation with syntactic changes that will be helpful for showing rectangularity.[13]

Let V_{old} be the verifier postulated in [Theorem 4.1](#) and denote by V_{new} the new, smooth and rectangular, verifier. We start by describing the proofs expected by V_{new}.

---

[13]Our presentation is from the "proof systems" perspective of PCP verifiers, rather than the "label cover" perspective given in [[DH13](#), Theorem 5.1]

Proofs in the new PCP system. New proofs are of length $2^r \cdot q$, which we think of as indexed over $\{0,1\}^r \times [q]$. Each location in the new proof corresponds to a full configuration of the original verifier. A correct proof in the original proof system is transformed to a correct proof in the new one by writing in the (R,k)-th location the answer of the original correct proof to the k-th query when randomness R is sampled. That is, for an input $x \in L$ and a correct proof π for V_{old} (i.e., one that is accepted w.p. 1), the (R,k)-th location of the correct proof for V_{new} will have the answer of π to the k-th query issued by V_{old} upon sampling random coin sequence R. Notice that in a correct proof for the new verifier, any two locations in the new proof corresponding to neighboring configurations (see Definition 2.8) should take the same value.

The new verifier. The basic idea is for the new verifier to emulate the original one: when the original samples coin sequence R, the new one queries locations $(R,1),\ldots,(R,q)$ in the new proof. However, if the original verifier queried the same location $i \in [\ell]$ for two different (neighboring) configurations (R,k) and (R',k'), a new ("cheating") proof could be *inconsistent* in its answers, using this inconsistency to cause the new verifier to accept when the original would not.

Thus, consistency between neighboring configurations must be checked. To guarantee smoothness and preserve the randomness of the new verifier, consistency is checked only between certain neighboring configurations, and not all. Namely, neighboring configurations are connected by a $O(\mu/q)$-sampler graph of constant degree, with edges on the sampler corresponding to consistency tests. Two configurations that are adjacent on the sampler are said to be sampler-neighbors, which is a stronger condition than being neighbors as per Definition 2.8.

In fact, both consistency and the original PCP verification are done in one fell swoop: when emulating the original verifier, the new verifier replaces a query to (R,k) with queries to its sampler-neighborhood, and checks its consistency. A sampler graph guarantees that inconsistency between neighboring configurations is reflected by this test w.h.p., so severely inconsistent proofs are rejected by the new verifier. On the other hand, regularity of the sampler implies smoothness, and its constant degree incurs only a constant blowup to the number of queries.

The point of this theorem is in showing rectangularity of the new verifier. Specifically, we ought to show how construction of the sampler and sampler-neighborhoods can be done *rectangularly*. That is, for any location $(R_{\text{row}}, R_{\text{col}}, R_{\text{shared}}, k)$, it is not enough to find all other sampler-neighboring $(R'_{\text{row}}, R'_{\text{col}}, R'_{\text{shared}}, k')$; it should be the case that the *row-part* of the sampler-neighbors can be found based on $(R_{\text{row}}, R_{\text{shared}}, k)$. Similarly, the *column-part* of the sampler-neighbors should be found only from $(R_{\text{col}}, R_{\text{shared}}, k)$.

We clarify what we mean by "row-part" and "column-part" of a query. The new proof (of length $2^r \cdot q$) can be thought of as a square matrix as follows: Fix a location $(R_{\text{row}}, R_{\text{col}}, R_{\text{shared.row}}, R_{\text{shared.col}}, k)$ in the new proof. Split k into k_{row} and k_{col}. The rows of the matrix are indexed by $(R_{\text{row}}, R_{\text{shared.row}}, k_{\text{row}})$, and the columns are indexed by $(R_{\text{col}}, R_{\text{shared.col}}, k_{\text{col}})$. Indeed, with this definitions, it is possible to find the row-parts (resp., column-parts) of the sampler-neighbors based on $(R_{\text{row}}, R_{\text{shared}}, k)$ (resp., $(R_{\text{row}}, R_{\text{shared}}, k)$) thanks to RNL.

Following is a detailed description of this construction.

Algorithm 4.4. Fix the original verifier V_{old}.

1. Sample a coin sequence R.

2. For each $k \in [q]$, construct the sampler of the neighborhood of (R,k) and check consistency of its sampler-neighborhood *in a rectangular way* as follows: Denote the randomness partition by of R by $(R_{\text{row}}, R_{\text{col}}, R_{\text{shared}})$.

   (a) Find the "row parts": Compute $L_{\text{row}} := A_{\text{row}}(R_{\text{row}}, R_{\text{shared}}, k)$ where A_{row} is the neighbor listing agent. Construct a canonical $(\mu/2q)$-sampler on the set of $|L_{\text{row}}|$ vertices, one corresponding to every entry in the list L_{row}. From the index of (R,k) in the list L_{row}, find the indices of the sampler-neighbors of (R,k), and output their "row-part" $(R'_{\text{row}}, R'_{\text{shared.row}}, k'_{\text{row}})$. In addition, output the row-part of (R,k).

   (b) Find the "column part": Similarly, compute $L_{\text{col}} := A_{\text{col}}(R_{\text{col}}, R_{\text{shared}}, k)$, construct a canonical $(\mu/2q)$-sampler on the set of $|L_{\text{col}}|$ vertices, find the indices of the sampler-neighbors of (R,k), and output their "column-part" $(R'_{\text{col}}, R'_{\text{shared.col}}, k'_{\text{col}})$. In addition, output the column-part of (R,k).

3. Let $\Delta - 1$ denote the degree of the sampler above. Note that we are making $\Delta \cdot q$ many queries. Feed the $\Delta \cdot q$ bits queried from the proof π_{new} to a circuit that first checks consistency between every sampler-neighborhood. That is, it check that in each of the q blocks of Δ bits, all the Δ bits are equal. If an inconsistency is spotted, the circuit immediately rejects. Otherwise, feed the first bit in every block to the decision circuit of the original verifier V_{old} (along with the p parity-checks on the randomness) and output its answer.

4.2 Proof of Theorem 4.1

Rectangularity. The randomness of the new verifier is split exactly the same as the original verifier into R_{row}, R_{col} and $R_{\text{shared}} = (R_{\text{shared.row}}, R_{\text{shared.col}})$. It follows from the description of the algorithm that R_{row} and R_{shared} determine the row-part of each of the $q \cdot \Delta$ queries. Similarly, R_{col} and R_{shared} determine the column-part of each of the $q \cdot \Delta$ queries.

ROP. The new decision predicate can be implemented by taking a circuit that checks equality on each of the q sampler-neighborhoods constructed in Item 2, and ANDing its answer with the output of original decision circuit (fed an arbitrary representative of each sampler-neighborhood, as well as the randomness parity checks). Therefore, the τ-ROP is preserved.

Query and decision complexities. By Fact 4.3, the size of each sampler-neighborhood (in a $(\mu/2q)$-sampler) is $\mathsf{poly}(q/\mu)$. A sampler-neighborhood is queried for each of the q original, so the query complexity (q/μ). As described in the ROP analysis, the new decision circuit can be obtained by ANDing the original decision circuit (of size d) to $\mathsf{poly}(q/\mu)$ equality checks. Thus, the size of the new circuit is $d + \mathsf{poly}(q/\mu)$.

Smoothness. The fact that the samplers we construct are Δ-regular graphs implies that every location in the new proof is read with exactly the same probability (see Appendix A.2 for more details).

Soundness. One way to see soundness (as well as smoothness) would be to observe Algorithm 4.4 is the same as the verifier of [DH13, Theorem 5.1], and is therefore sound (and smooth). Since the latter theorem and its proof are described in the "label cover" view of PCPs whereas our work takes the "proof systems" view, we present an alternative proof in the latter view in Appendix A.1.

Runtime complexity. The new verifier emulates the original one. In addition, for each of the q queries it invokes RNL agents and finds a neighborhood in the sampler. Invoking RNL agents takes t_{RNL} time. Constructing the explicit sampler on the configuration's neighborhood and finding its sampler-neighborhood takes time at most $\mathsf{poly}(t_{\text{RNL}})$ time (we upper bound the size of each list with the runtime of each agent).

5 A many-query robust PCP with RNL

In this section, we prove that the Reed–Muller-based PCP of Ben-Sasson *et al.* [BGHSV06, BGHSV05] has RNL. This PCP issues many queries, but is *robust* – which will come useful later in the composition stage (Section 7) to reduce its query complexity. In particular, we modify the many-query robust PCP of Ben-Sasson *et al.* [BGHSV06, BGHSV05] to obtain the following PCP with RNL.

Theorem 5.1 (Strengthening of [BGHSV06, Theorem 3.1] and [BGHSV05], simplified[14]). *Suppose that L is a language in $\mathsf{NTIME}(T(n))$ for some non-decreasing function $T \colon \mathbb{N} \to \mathbb{N}$. There exists a universal constant c such for all odd integers $m \in \mathbb{N}$ and $s \in (0, 1/2)$ satisfying $T(n)^{1/m} \geq m^{cm}/s^6$, L has a robust PCP with the following parameters:*

1. *Alphabet $\{0, 1\}$.*

2. *Randomness complexity $r(n) = (1 - \frac{1}{m}) \log T(n) + O(m \log \log T(n)) + O(\log(1/s))$.*

3. *Decision and Query complexity $d(n) = q(n) = T(n)^{1/m} \cdot \mathsf{poly}(\log T(n), 1/s)$.*

17

4. *Robust soundness error s with robustness parameter $\Theta(s)$.*

5. *Runtime complexity $t(n) = q(n) \cdot \mathsf{poly}(n, \log T(n))$.*

6. *The PCP verifier has τ-RNL with running time $t_{\mathrm{RNL}}(n)$ where*

$$\tau \cdot r(n) = r_{\mathrm{shared}} = \frac{4}{m} \log T(n) + O(m \log \log T(n)) + O(\log(1/s)),$$
$$t_{\mathrm{RNL}}(n) = \mathsf{poly}(\log T(n)).$$

Remark 5.2. *Items 1 to 4 are exactly as in the statement of [BGHSV06, Theorem 3.1], the outer robust PCP construction of Ben-Sasson et al. while Item 5 (the verifier running time) is obtained by the efficient PCP verifiers of Ben-Sasson et al. [BGHSV05]. The main difference between the two works of Ben-Sasson et al. [BGHSV06, BGHSV05] is that the latter uses a reduction from Succinct-SAT to Succinct-Multivariate-Algebraic-CSP. To show that these PCPs have RNL, we need to analyze the query and predicate of the corresponding PCP verifiers. Since these are almost identical in both the constructions (i.e., the original robust PCP construction and the subsequent efficient version of it), we work with the robust PCP of [BGHSV06] and just observe that these modifications can be carried out efficiently as in [BGHSV05].*[15]

The robust PCP verifiers of Ben-Sasson *et al.* [BGHSV06, BGHSV05] already have the properties listed in Items 1 to 5. In this section we show that it also has RNL as stated in Item 6 above. This is done in two steps: First (Section 5.1), we show that the robust PCP from [BGHSV06, Section 8.2.1] has RNL, albeit over a large alphabet. Second (Section 5.2), we reduce the alphabet size to binary while preserving RNL.

5.1 The robust RNL PCP verifier over a large alphabet

Lemma 5.3. *There exists a robust PCP verifier for a language in $\mathsf{NTIME}(T(n))$ with the properties mentioned in Theorem 5.1, but over an alphabet of size $2^{\mathsf{polylog}(T(n))}$ (instead of the Boolean alphabet).*

For convenience, we remind the reader of the PCP verifier from [BGHSV05], following its presentation in [BGHSV06, Section 8.2.1] (see Remark 5.5 for a minor difference). That same work shows completeness and robust soundness of this verifier. Thus, we need only to show that this PCP has RNL, so it suffices for us to recall its *query patterns* without detailing the way its *decision (predicate)* is made based on these queries.

Algorithm 5.4 (Verifier query pattern [BGHSV06, Section 8.2.1])**.** Let $\mathbb{F}$ be a field of size $|\mathbb{F}| = T(n)^{1/m} \cdot \mathsf{poly} \log T(n)$ where m is an odd integer. The proof oracle is a map $\Pi \colon \mathbb{F}^m \to \mathbb{F}^d$ where $d = m \cdot \mathsf{poly}(\log T(n))$. Let $\mathsf{shift} \colon \mathbb{F}^m \to \mathbb{F}^m$ denote the linear transformation that cyclically shifts each coordinate to the left; i.e., $\mathsf{shift}(x_1, \ldots, x_m) \coloneqq (x_2, x_3, \ldots, x_m, x_1)$.

The queries will be based on *lines* through $\mathbb{F}^m$, and their shifts. Recall that the line $\mathcal{L}$ with intercept $x \in \mathbb{F}^m$ and direction $y \in \mathbb{F}^m$ is the set $\mathcal{L} \coloneqq \{x + ty \colon t \in \mathbb{F}\}$. The verifier samples lines from two distributions:

- A first-axis parallel line, in which $y = (1, 0, \ldots, 0)$, and x is sampled uniformly at random from $\{0\} \times \mathbb{F}^{m-1}$.

- A canonical pseudorandom line is chosen by first uniformly sampling a direction y from a λ-biased set $S_\lambda \subseteq \mathbb{F}^m$. The direction y partitions the space $\mathbb{F}^m$ into $|\mathbb{F}|^{m-1}$ parallel lines, and one of these lines is chosen uniformly at random. Note that this requires $\log(|S_\lambda|) + (m-1) \log(|\mathbb{F}|)$ random bits. Without loss of generality, we assume that all $y \in S_\lambda$ have $y_1 \neq 0$.[16] Thus, similarly to first-axis parallel lines, x can be sampled uniformly at random from $\{0\} \times \mathbb{F}^{m-1}$.

Remark 5.5 (Differences in the query pattern of [BGHSV06] and [BGHSV05])**.** *The only difference in the robust PCP construction of [BGHSV06] and its efficient counterpart in [BGHSV05] is the reduction from $\mathsf{NTIME}(T(N))$ to (succinct) Multivariate-Algebraic-CSP (which in turn uses the Pippenger–Fisher reduction [PF79]) [BGHSV05, Definition 6.3, Theorem 6.4]. This causes the field size to increase from $O(m^2 \cdot T(n)^{1/m})$ to $T(n)^{1/m} \cdot \mathsf{poly} \log T(n)$.*

---

[15]For the sake of simplicity, instead of working with verifier specifications (as in we work with verifiers instead of verifier specifications ([BGHSV05] use the latter). This costs us an extra $q(n)$ factor in the running time. However this loss of $q(n)$ has no effect on our application to rigid matrices.

[16]To obtain such a λ-biased set, construct a $\lambda/2$-biased set and remove all y's with $y_1 = 0$ from it.
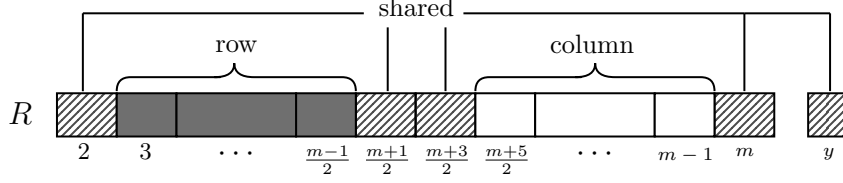
Figure 1: The partition of the randomness of Algorithm 5.4.

Proof of Lemma 5.3. We prove that the verifier of Algorithm 5.4 has RNL. First, note that the verifier uses a total of $\log(|S_\lambda|) + (m-1)\log(|\mathbb{F}|)$ random bits to sample a first-axis parallel line $\mathcal{L}_0$ and a canonical pseudorandom line $\mathcal{L}_1$, reusing the random bits between these two lines. It then queries the proof oracle on $\mathcal{L}_0$, $\mathsf{shift}(\mathcal{L}_0)$, $\mathcal{L}_1$, and $\mathsf{shift}(\mathcal{L}_1)$.[17]

The randomness used for sampling x is partitioned into $(m-1)$ parts of equal length, denoted by $(R_2, R_3, \ldots, R_m)$ where $|R_i| = \log(|\mathbb{F}|)$, and R_i determines x_i for each $i \in \{2, \ldots, m\}$ (recall that $x_1 = 0$ always). The randomness used to sample a direction from S_λ is denoted by R_y. Recall that there are two-types of lines, canonical and axis-parallel lines. In both cases the direction of the line y is a function of the bits R_y (in the axis-parallel line the direction is just the constant function).

The row, column and shared parts of the randomness are portrayed in Fig. 1. Formally,

$$R_{\mathrm{row}} := \big(R_3, \ldots, R_{(m-1)/2}\big)$$
$$R_{\mathrm{col}} := \big(R_{(m+5)/2}, \ldots, R_{m-1}\big)$$
$$R_{\mathrm{shared.row}} := (R_2, R_{(m+1)/2})$$
$$R_{\mathrm{shared.col}} := (R_{(m+3)/2}, R_m)$$
$$R_{\mathrm{shared}} := (R_{\mathrm{shared.row}}, R_{\mathrm{shared.col}}, R_y),$$

where R_y is divided arbitrarily between $R_{\mathrm{shared.row}}$ and $R_{\mathrm{shared.col}}$. Thus, the randomness has parts of length $r_{\mathrm{row}} := |R_{\mathrm{row}}|$, $r_{\mathrm{col}} := |R_{\mathrm{col}}|$, $r_{\mathrm{shared}} := |R_{\mathrm{shared}}|$, with total randomness $r = r_{\mathrm{row}} + r_{\mathrm{col}} + r_{\mathrm{shared}}$ and $\tau \cdot r = r_{\mathrm{shared}}$.

Rectangular Neighbor-Listing (RNL) The BGHSV verifier makes $4 \cdot |\mathbb{F}|$ queries. We index the queries by $k := (b_1, b_2, t) \in \{0,1\}^2 \times \mathbb{F}$ as follows.

- $b_1 = 0$ indicates the query is to a line. $b_1 = 1$ indicates it is to a shifted line.

- $b_2 = 0$ indicates the query is to a first-axis parallel line. $b_2 = 1$ indicates it is to a canonical line.

- $t \in \mathbb{F}$ indicates the position on the line.

Given a configuration (R, k) that results in a query to location $z \in \mathbb{F}^m$, we ought to show how to list all neighboring configurations (i.e. configurations that lead to location z) in a rectangular way. We first show who are the neighboring configurations, and then show how to rectangularly (and synchronously) list them by describing the listing agents A_{row} and A_{col}. This shows that the verifier has RNL.

Neighbors of (R, k). A full configuration $(R, k) = (R_{\mathrm{row}}, R_{\mathrm{col}}, R_{\mathrm{shared}}, k)$ specifies $k = (b_1, b_2, t)$, $x = (0, R_2, \ldots, R_m)$, and $R_y \in [|S_\lambda|]$. R_y and b_2 determine $y \in \mathbb{F}^m$ as follows: if $b_2 = 0$ then $y = (1, 0, \ldots, 0)$, otherwise $y = S_\lambda[R_y]$. Recall that $\mathsf{shift}(x)$ denotes cyclic shift of x one step to the left. Given the full configuration specified by $k = (b_1, b_2, t)$, x and R_y, the location of the kth query will be $x + t \cdot y$ if $b_1 = 0$, or $\mathsf{shift}(x + t \cdot y)$ if $b_1 = 1$. More concisely, letting $\mathsf{shift}_j(x)$ denote the cyclic shift of x by j steps for any $j \in \mathbb{Z}$, the location of the kth query is $\mathsf{shift}_{b_1}(x + t \cdot y)$.

Thus, any other configuration (R', k') that specifies $k' = (b_1', b_2', t')$, x' and R_y' would query the same location if and only if

$$\mathsf{shift}_{b_1'}(x' + t' \cdot y') = \mathsf{shift}_{b_1}(x + t \cdot y). \tag{2}$$

---

[17]The size of the set S_λ and a detailed description of how we derived this query pattern based on the tests of the verifier of [BGHSV06, Section 8.2.1] can be found in Appendix C.

19

In other words, (R', k') neighbors (R, k) if and only if it satisfies Eq. (2).

The neighbor listing agents. Rearranging Eq. (2), we see that $(x, b_1, b'_1, t, t', y, y')$ uniquely determines x' by the equation

$$x' = \mathsf{shift}_{b_1 - b'_1}(x + t \cdot y) - t' \cdot y', \tag{3}$$

Recall that any location queried must fulfill the condition $x'_1 = 0$. For any possible $k' = (b'_1, b'_2, t')$ and y', fulfillment of this condition is determined only by $k = (b_1, b_2, t)$, y and (x_2, x_m). Thus, we say that partial configuration (R'_y, k') is realizable (for (R_{shared}, k)) if any only if $x'_1 = 0$.

Furthermore, notice that for any (R'_y, k') there is a unique t' such that Eq. (3) has $x'_1 = 0$, and finding such t' can be done in a constant number of basic arithmetic operations. This will come in handy shortly, when we construct the listing agents.

We can now present two algorithms listing all neighbors of a given configuration in a rectangular and synchronized fashion. First the row neighbor-listing agent A_{row}. On input row configuration $(R_{\mathrm{row}}, R_{\mathrm{shared}}, k)$,

1. Obtain t, b_1, b_2, y, $x_{[2,(m+3)/2]}$ and x_m from the input. Initialize an empty list L_{row}.

2. For each (R'_y, b'_1, b'_2):

   (a) Find $t' \in \mathbb{F}$ such that $(R'_y, k') = (t', b'_1, b'_2)$ is realizable (as previously explained).

   (b) Compute $x'_{\mathrm{row}} := x'_{[2,(m+1)/2]}$ as in Eq. (3). Append $(x'_{\mathrm{row}}, R'_y, k')$ to the list L_{row}.

3. Sort L_{row} according to (R'_y, k'). Output L_{row}.

The column neighbor-listing agent runs similarly. On input column configuration $(R_{\mathrm{col}}, R_{\mathrm{shared}}, k)$,

1. Obtain t, b_1, b_2, y, x_2 and $x_{[(m+1)/2,m]}$ from the input. Initialize an empty list L_{col}.

2. For each (R'_y, b'_1, b'_2):

   (a) Find $t' \in \mathbb{F}$ such that $(R'_y, k') = (t', b'_1, b'_2)$ is realizable (as previously explained).

   (b) Compute $x'_{\mathrm{col}} := x'_{[(m+3)/2,m]}$ as in Eq. (3). Append $(x'_{\mathrm{col}}, R'_y, k')$ to the list L_{col}.

3. Sort L_{col} according to (R'_y, k'). Output L_{col}.

Note that both agents give lists indexed and sorted by all realizable (R'_y, k'). Thus, both arrays are of the same length and ordering. Moreover, the ordering of L_{row} and L_{col} when the agents are input any two neighboring configurations (R, k) and (R', k'). Thus, Item 2 of Definition 2.9 is satisfied.

For each $i \in [|L_{\mathrm{row}}|]$, concatenating $L_{\mathrm{row}}[i]$ and $L_{\mathrm{col}}[i]$ gives a tuple $(x'_{\mathrm{row}}, R'_y, k', x'_{\mathrm{col}}, R'_y, k')$. By splitting R'_y arbitrarily into $R'_{y.\mathrm{row}}$ and $R'_{y.\mathrm{col}}$ we have that L_{row} and L_{col}, when appropriately "zipped" (as in Item 1 of Definition 2.9) give the list L of all neighboring configurations to (R, k).

To see Item 3 of Definition 2.9, note that both A_{row} and A_{col} can identify the index of (R, k) in L since it corresponds to the index of the entry whose two last elements are (R_y, k) in L_{row} and in L_{col}.

Lastly, we calculate t_{RNL}, which is the runtime of A_{row} (the case of A_{col} is analogous). Item 1 takes $\mathsf{poly}(m \cdot \log |\mathbb{F}| \cdot \log(1/s))$ time; this includes getting $y \in S_\lambda$ from a random string R_y using the efficient construction of λ-biased set in [AGHP92, NN93]. For each (R'_y, b'_1, b'_2), Items 2a and 2b each take $O(m \cdot \log |\mathbb{F}|)$ time. The output list length is upper bounded by $4|S_\lambda| = \mathsf{poly}(m \cdot \log |\mathbb{F}| \cdot \log(1/s))$, so sorting it in Item 3 takes at most $\mathsf{poly}(m \cdot \log |\mathbb{F}| \cdot \log(1/s))$ time. Thus, overall the running time is dominated by $\mathsf{poly}(m \cdot \log |\mathbb{F}| \cdot \log(1/s))$. By the choice of parameters in Theorem 5.1 and size of $\mathbb{F}$, this is asymptotically equal to $\mathsf{poly}(\log T(n))$.

$\square$

5.2 Alphabet Reduction

Lemma 5.3 gives a robust PCP with RNL *over a large alphabet*, but the final construct requires a PCP over the *Boolean alphabet*. Next, we show that standard alphabet reduction [For65] preserves RNL. Namely, each symbol is replaced with its encoding in a binary error correcting code. We use the following lemma which gives a constant rate and constant distance code such that the decoding and encoding time is linear.

Theorem 5.6 ([Spi96]). *There is a constant rate, constant distance linear error correcting code with a linear-time encoder, and a linear-time decoder recovering a message from a codeword with up to a fixed constant fraction of errors.*

Lemma 5.7. *Suppose language L has a PCP with verifier V as described in Table 2, then L has a PCP with verifier V' as described in Table 2. Furthermore, if V has τ-RNL, then so does V'.*

Complexity	V	V'
Alphabet	Σ	$\{0, 1\}$
Robust soundness error	s	s
Robustness parameter	ρ	$\Omega(\rho)$
Randomness	r	r
Query	q	$O(q \cdot \log |\Sigma|)$
Proof length	m	$O(m \cdot \log |\Sigma|)$
Decision	d	$d \cdot \mathsf{polylog}(|\Sigma|)$
Runtime	t	$t \cdot \mathsf{polylog}(|\Sigma|)$

Table 2: The complexities of original verifier V and the Boolean verifier V'.

Proof. Fix a linear-time computable and decodable error correcting code of constant rate and distance, denoted $\mathsf{Enc} \colon \Sigma \to \{0, 1\}^\sigma$ for $\sigma = O(\log |\Sigma|)$ from Theorem 5.6. Assume that the first $\log |\Sigma|$ bits in the encoding are the binary representation of the (non-binary) message. The new (Boolean) proof is written in a natural way: each non-binary symbol is replaced with its encoding under Enc.

The Boolean PCP verifier V' emulates the non-Boolean verifier V as follows: when the non-Boolean verifier queries a location $b \in [m]$, the Boolean verifier queries the whole block $[(b-1)\sigma + 1, b\sigma]$ and checks if it is a valid encoding of the symbol specified by the first $\log |\Sigma|$ bits (if not, reject). Once all the queries are decoded correctly, the verifier V' does the verification on the decoded values as V.

It is easy to observe that the query complexity and the proof length increase by a factor of $O(\log |\Sigma|)$. Since the new verifier has to perform the decoding of an error correcting code, this adds a multiplicative overhead of $\mathsf{polylog}(|\Sigma|)$ in the running time and the decision complexity.

The importance of using the error correcting code is to make sure that the new verifier V' is still robust. This follows from [BGHSV06, Lemma 2.13], where it was shown that the soundness error remains the same and the robustness parameter decreases by a constant factor.

It is also easy to observe that RNL is preserved with the same partition of the randomness. Fix a j-th location from the block $[(b-1)\sigma + 1, b\sigma]$. First observe the following proposition:

Proposition (a): If a full configuration (R, k) queries a location $b \in [m]$ in the original proof, then the configuration $(R, \sigma(k-1) + j)$ queries the j-th location from the block $[(b-1)\sigma + 1, b\sigma]$ in the new proof and vice-versa.

Let A_{row} and A_{col} are the row and column agents of the non-Boolean verifier V. The row agent $A'_{\mathrm{row}}(R_{\mathrm{row}}, R_{\mathrm{shared.row}}, \tilde{k}) \to L'_{\mathrm{row}}$ and the column agent $A'_{\mathrm{col}}(R_{\mathrm{col}}, R_{\mathrm{shared.col}}, \tilde{k}) \to L'_{\mathrm{col}}$ of the new verifier V' are as follows: Both the agents first compute the block number $k = \lceil \tilde{k}/\sigma \rceil$ and the index $j = \tilde{k} - \sigma(k-1)$.

$$L'_{\mathrm{row}}[i] := (R'_{\mathrm{row}}, R'_{\mathrm{shared.row}}, \sigma(k'-1) + j) \quad s.t. \quad (R'_{\mathrm{row}}, R'_{\mathrm{shared.row}}, k') = L_{\mathrm{row}}[i]$$
$$L'_{\mathrm{col}}[i] := (R'_{\mathrm{col}}, R'_{\mathrm{shared.col}}, \sigma(k'-1) + j) \quad s.t. \quad (R'_{\mathrm{col}}, R'_{\mathrm{shared.col}}, k') = L_{\mathrm{col}}[i]$$

where $L_{\mathrm{row}} \leftarrow A_{\mathrm{row}}(R_{\mathrm{row}}, R_{\mathrm{shared.row}}, k)$ and $L_{\mathrm{col}} \leftarrow A_{\mathrm{col}}(R_{\mathrm{col}}, R_{\mathrm{shared.col}}, k)$. To see the correctness, suppose the full configuration $(R_{\mathrm{row}}, R_{\mathrm{col}}, R_{\mathrm{shared.row}}, R_{\mathrm{shared.col}}, k)$ queries the location $b \in [m]$ from the original

proof. By RNL of V, the full configuration $(R'_{\text{row}}, R'_{\text{col}}, R'_{\text{shared.row}}, R'_{\text{shared.col}}, k')$ given by $L_{\text{row}}[i]$ and $L_{\text{col}}[i]$ leads to the same location $b \in [m]$. Using Proposition (a), we can conclude that the verifier V' when given the full configuration $(R'_{\text{row}}, R'_{\text{col}}, R'_{\text{shared.row}}, R'_{\text{shared.col}}, \sigma(k'-1)+j)$, queries the location $(b-1)\sigma+j$ in the new proof. As $\tilde{k} = \sigma(k-1)+j$, again using Proposition (a), the input full configuration to the agents $(R'_{\text{row}}, R'_{\text{col}}, R'_{\text{shared.row}}, R'_{\text{shared.col}}, \tilde{k})$ also leads to the same location $(b-1)\sigma+j$ in the new proof. Therefore, every full configuration given by $(L'_{\text{row}}[i], L'_{\text{col}}[i])$ leads to the location $(b-1)\sigma+j$ in the new proof. Furthermore, since the number of full configurations on which V queries $b \in [m]$ is the same as the number of full configurations on which V' queries any fixed location from the block $[(b-1)\sigma+1, b\sigma]$ (in fact, there is a bijection given by Proposition (a)), the list $(L'_{\text{row}}, L'_{\text{col}})$ is exhaustive. $\qquad\square$

We now finish the proof of Theorem 5.1.

Proof of Theorem 5.1. Lemma 5.3 shows the existence of a verifier with the additional τ-RNL over a large alphabet. The alphabet reduction technique from Lemma 5.7 converts the PCP to a PCP over the Boolean alphabet. Since, the original PCP is over an alphabet of size $2^{\mathsf{polylog}(T(n))}$, this conversion increases the proof length, query complexity, decision complexity and the verifier's running time by a multiplicative factor of $\mathsf{polylog}(T(n))$. With all these changes, these four parameters of the new verifier are asymptotically same as the ones mentioned in Theorem 5.1.

In the whole process, the robustness parameter of the verifier changes by a constant multiplicative factor and this change is irrelevant in proving the lemma. $\qquad\square$

6 Adding ROP to a robust PCP

One way at looking at the verification procedure is as follows: On sampling the randomness R, the verifier constructs a circuit $D := D(R)$ and a subset $I := I(R)$ of proof locations of size q. The verifier outputs the verdict of $D(\pi|_I)$. In this abstract way, the circuit D depends on the full randomness R. However, for our application we need the verifier to have randomness-oblivious circuits (ROP).

Recall that the ROP states that the decision predicate depends only on a small fraction of the randomness, but may take as input a limited number of parity checks on the entire randomness. We generalize robust soundness to the ROP setting in the natural way, measuring the distance of both the bits read by the verifier *as well as the randomness parity checks* from satisfying the decision predicate. This definition will be useful when we compose a robust PCP having ROP with a PCPP (in Section 7).

Definition 6.1 (robust soundness for ROP verifier)**.** *For functions $s, \rho : \mathbb{N} \to [0,1]$, a PCP verifier V for a language L with τ-ROP and parity check complexity p has* robust-soundness error s *with* robustness parameter ρ *if the following holds for every $x \notin L$: For every oracle π, the input to the decision predicate (that consists of bits read by the verifier and parities of the randomness) are ρ-close to being accepted with probability strictly less than s. Formally,*

$$\forall \pi \quad \Pr_{(I,D,P) \xleftarrow{R} V(x)} [\exists a, b \text{ s.t. } D(ab) = 1 \text{ and } \delta(ab, \pi|_I P) \le \rho] < s(|x|).$$

Lemma 6.2. *Suppose language L has a PCP with verifier V as described in Table 3, then L has a PCP with verifier V' as described in Table 3 with 0-ROP. Furthermore, if V has τ-RNL, then so does V'.*

Proof. We can replace the circuit D of V, which depends on the randomness R, with another circuit D_{ROP} such that $D(x) = D_{\text{ROP}}(x, R)$ for all $x \in \{0,1\}^q$. In other words, we give R as an explicit input to the circuit D_{ROP} and therefore remove the dependence of D_{ROP} on R. Note however that the output of D_{ROP} depends on both the randomness R as well as the proof locations $\pi|_I$.

It is easy to see that this preserves the completeness and soundness of the PCP. However, this transformation might lose the guarantee on the *robust soundness* when we look at the input to the circuit. This is because even if $\pi|_I$ is far from satisfying D, it might be the case that $(\pi|_I, R)$ is close to satisfying D' (in this case when measuring the distance from satisfying answers, changes in R are also allowed).

In order to overcome this issue we encode the randomness with a good error code. By suitably repeating the code of Theorem 5.6, we have a linear error correcting code $\text{Enc}: \{0,1\}^r \to \{0,1\}^q$ with constant relative distance. Based on its linear time decoder, two circuits are constructed:

Complexity	V	V'
Robust soundness error	s	s
Robustness parameter	ρ	$\Omega(\rho)$
Randomness	r	r
Query	$q \geq r$	q
Proof length	m	m
Decision	d	$\widetilde{O}(t)$
Parity-check	$-$	q
Runtime	t	$\widetilde{O}(t)$

Table 3: The complexities of the original verifier V and the 0-ROP verifier V'.

- Circuit Dec that on input $y \in \{0,1\}^q$ outputs $x \in \{0,1\}^r$ such that $\mathrm{Enc}(x) = y$ if such x exists, and an arbitrary value otherwise.[18]

- Circuit Test that on input $y \in \{0,1\}^q$ outputs 1 if y is a codeword (i.e., in the image of Enc), and 0 if it is not.

The decision circuit generated by V' is $D'(z,y) = D_{\mathrm{ROP}}(z, \mathrm{Dec}(y)) \wedge \mathrm{Test}(y)$. The parity checks generated by V' are simply $\mathrm{Enc}(R)$ – indeed, since the encoding is linear then $\mathrm{Enc}(R)_1, \ldots, \mathrm{Enc}(R)_q$ are linear functions in R.

Decision and runtime complexities. The original verifier V constructs D in time t. The circuit D_{ROP} can emulate this using $\widetilde{O}(t)$ gates [Coo88]. By Theorem 5.6, the runtime of decoding and testing is $O(q)$, thus circuits Dec and Test can be constructed using at most $\widetilde{O}(q) = \widetilde{O}(t)$ gates. All in all, the size of D' is at most $\widetilde{O}(t)$.

Apart from constructing D', the verifier V' also produces the queries of V and the parity checks $\mathrm{Enc}(R)$. Queries are constructed in time at most t, and again by Theorem 5.6 computing $\mathrm{Enc}(R)$ takes time $O(q)$. Thus the runtime of V' is at most $\widetilde{O}(t)$.

Robust soundness. The original verifier has robust-soundness error s with robustness parameter ρ, so with probability at least $1 - s$ the input $\pi|_I$ to the circuit $D := D(R)$ is at least ρ-far from satisfying D. This means that with probability at least $1 - s$, $(\pi|_I, E(R))$ is at least $\min\{\rho/2, \Omega(1)\}$ far from satisfying D'. To see that, note that to satisfy D' either the first half of the input $\pi|_I$ needs to be changed in $q \cdot \rho$ locations, or the second half of the input $E(R)$ needs to be changed to another legal encoding $E(R')$ which by the properties of E requires $\Omega(q)$ bit-changes. Thus, the robustness parameter of V' is $\min\{\rho/2, \Omega(1)\} \geq \Omega(\rho)$, and the robust soundness error remains s. $\qquad\square$

7 RNL-preserving PCP composition

In this section, we strengthen the composition theorem of [BGHSV06] by showing that it preserves RNL and (to some extent) ROP of a robust PCP.

7.1 PCPs of Proximity

Recall that we think of a PCP verifier as accepting input x and proof π if the answers received from π, denoted $\pi|_I$, satisfy the decision circuit D generated by V. In a nutshell, PCP composition is done by replacing the naive verification of the claim *"$\pi|_I$ satisfies D"* with a verification by an inner verifier.

The goal of PCP composition is to reduce the query complexity of an (outer) verifier by composing it with an inner verifier of smaller (even constant) query complexity. Hence, the inner verifier has restricted access not only to its proof, but also to part of its input (namely, $\pi|_I$). Since such a constrained verifier cannot

---

[18]In fact, the circuit only needs decode valid codewords, which is easier than decoding noisy codewords.

distinguish between answers that satisfy D to those that are close to satisfying D, its soundness condition is relaxed to rejection of answers that are *far* from all satisfying assignments to D. Indeed, if the outer PCP had suitable robustness, this relaxation still yields a sound PCP. We formalize this discussion next.

Definition 7.1 (Pair language and CVP). *A* pair language L *is a subset of* $\{0,1\}^* \times \{0,1\}^*$. *The* Circuit Valuation Problem *(*CVP*) is the pair language consisting of circuits and their accepting inputs. Formally,*

$$\mathsf{CVP} := \{(C, y) | C(y) = 1\}.$$

A PCP of Proximity for a pair language L is given a pair (x, y) and a proof π, where access to x is *explicit* (i.e., x can be read entirely) while only oracle access is given to y and π (so queries to y are accounted for in the verifier's query complexity). The soundness condition is weakened to rejection with high probability only of (x, y) such that y is *far* from $L(x) := \{y' | (x, y') \in L\}$. Formally:

Definition 7.2 (PCP of Proximity (PCPP)). *Let* $L \subseteq \{0,1\}^* \times \{0,1\}^*$ *be a pair language. For* $s, \delta \colon \mathbb{N} \to \mathbb{N}$, *a restricted verifier* V *over* Σ *is a* PCP of proximity verifier *for* L *with* proximity parameter δ *and* soundness error s *if the following two conditions hold for any* $x, y \in \{0,1\}^*$:

Completeness *If* $(x, y) \in L$, *then there exists a proof* π *such that* $V(x)$ *accepts the oracle* $y\pi$ *(*y *is called the* input oracle *and* π *is called the* proof oracle*) with probability* 1. *Formally,*

$$\exists \pi \qquad \Pr_{(I,D) \xleftarrow{R} V(x)} [D(y\pi|_I) = 1] = 1.$$

Soundness *If* y *is* δ*-far from the set* $L(x) = \{y' \mid (x, y') \in L\}$, *then for every proof oracle* π, $V(x)$ *accepts the oracle* $y\pi$ *with probability strictly less than* s. *Formally,*

$$\forall \pi \qquad \Pr_{(I,D) \xleftarrow{R} V(x)} [D(y\pi|_I) = 1] < s(|x|).$$

For convenience, we assume that the input locations $I = (i_1, \ldots, i_q)$ *are each of the form* $i_k = (b_k, j_k)$, *where* b_k *is a bit signifying the oracle of the* k*-th query, and* j_k *is the location in that oracle. Formally, for each* $k \in [q]$, *if* $b_k = 0$ *(resp.* $b_k = 1$*) then* $j_k \in [|y|]$ *(resp.* $j_k \in [|\pi|]$*) is the location in* y *(resp. in* π*) of the* k*-th query of* V.

7.2 The composition theorem

Ben-Sasson *et al.* [BGHSV06] show that the composition of a robust PCP V_{out} and a PCP of proximity V_{in} with suitable parameters yields a sound composite PCP (described in Fig. 2) that enjoys the inner query complexity and (roughly) the outer randomness complexity.

Theorem 7.3 ([BGHSV06, Theorem 2.7]). *Suppose language* L *has a robust PCP verifier* V_{out}, *and that* CVP *has a* PCPP *verifier* V_{in}, *with parameters described in* Table 4 *such that* $\rho_{\text{out}} \geq \delta_{\text{in}}$. *Then, language* L *has a PCP verifier* V_{comp} *as described in* Table 4.

Our goal is to reduce the query complexity of the PCP of Section 5 by composing it with a constant-query inner PCPP (see Section 8). Towards this end, we adapt the composition to consider ROP (described in Fig. 3), and strengthen Theorem 7.3 by showing that it preserves RNL of the outer PCP, and (to some extent) its ROP.

Lemma 7.4. *In the setting of* Theorem 7.3, *assume further that* V_{out} *has* 0*-ROP with parity-check complexity* p_{out}, *and* τ*-RNL with listing agent runtime of* t_{RNL}. *Then* V_{comp} *has* $\hat{\tau}$*-ROP with parity-check complexity* p_{out}, *and* $\hat{\tau}$*-RNL with listing agent runtime of* $t_{\text{RNL}} \cdot 2^{r_{\text{in}}} \cdot q_{\text{in}} \cdot t_{\text{in}}$, *where*

$$\hat{\tau} = \frac{r_{\text{in}} + \tau \cdot r_{\text{out}}}{r_{\text{in}} + r_{\text{out}}}.$$

Furthermore, the shared and aware parts of the randomness are the same.

Hardwired: Outer verifier V_{out} and inner verifier V_{in}.

Input: Explicit input x, outer proof Π and inner proofs $\{\pi_{R_{\text{out}}} \mid R_{\text{out}} \in \{0,1\}^{r_{\text{out}}}\}$.

1. Sample $R_{\text{out}} \in \{0,1\}^{r_{\text{out}}}$.

2. Run $V_{\text{out}}(x; R_{\text{out}})$ to obtain $I_{\text{out}} = (i_1, i_2, \ldots, i_{q_{\text{out}}})$ and D_{out}.

3. Sample $R_{\text{in}} \in \{0,1\}^{r_{\text{in}}}$.

4. Run $V_{\text{in}}(D_{\text{out}}, R_{\text{in}})$ to obtain $I_{\text{in}} = ((b_1, j_1), \ldots, (b_{q_{\text{in}}}, j_{q_{\text{in}}}))$ and D_{in}.

5. Initialize $I_{\text{comp}}, P_{\text{comp}} := \emptyset$, and $D_{\text{comp}} := D_{\text{in}}$

6. For each $k \in [q_{\text{in}}]$, determine the queries of the composite verifier:

   (a) If $b_k = 0$, let $\hat{i}_k$ be the location of $\Pi(i_{j_k})$. Append $\hat{i}$ to I_{comp}.

   (b) If $b_k = 1$, let $\hat{i}_k$ be the location of $\pi_{R_{\text{out}}}(j_k)$. Append $\hat{i}$ to I_{comp}.

Output $(I_{\text{comp}}, D_{\text{comp}})$.

Figure 2: The composite verifier V_{comp} of [BGHSV06].

Complexity	V_{out}	V_{in}	V_{comp}
Soundness error	(Robust:) s_{out}	s_{in}	$s_{\text{out}} + s_{\text{in}}$
Proximity parameter	-	δ_{in}	-
Robustness parameter	ρ_{out}	-	-
Randomness	r_{out}	r_{in}	$r_{\text{out}} + r_{\text{in}}$
Query	q_{out}	q_{in}	q_{in}
Decision	d_{out}	d_{in}	d_{in}
Runtime	t_{out}	t_{in}	$t_{\text{out}} + t_{\text{in}}$

Table 4: The complexities of V_{out}, V_{in} and V_{comp}. The complexities of each verifier are taken with respect to its input; that is, the complexities of the outer and composite verifier are with respect to n, while those of the inner verifier are with respect to $d_{\text{out}}(n)$. For example, $r_{\text{out}} + r_{\text{in}}$ refers to $r_{\text{out}}(n) + r_{\text{in}}(d_{\text{out}}(n))$.

Remark 7.5. *We stress that the outer verifier's ROP (as well as RNL) is used in showing RNL of the composite verifier. Briefly, this is so that the composite listing agents can emulate the inner verifier so as to obtain its queries, without having access to the outer randomness.*

Proof. First, some names and notation. In the following proof, V_{out}, V_{in} and V_{comp} will be called the outer, inner and composite verifiers (respectively). We affix the terms outer, inner and composite when discussing components of the respective verifiers. For example, the outer randomness refers to the randomness used by the outer verifier. We denote the outer randomness by R_{out}, the inner randomness by R_{in}, and the composite by $\hat{R}$ – this will avoid double-indices when we further partition the outer and composite randomness.

Soundness   Note that the outer verifier V_{out} on randomness R_{out} queries the outer proof Π at q_{out} many locations I. The verifier then computes p_{out} many parities $P_{\text{out}} := (C_1(R_{\text{out}}), C_2(R_{\text{out}}), \ldots, C_{p_{\text{out}}}(R_{\text{out}}))$ of R_{out} and evaluates the circuit $D_{\text{out}}(\Pi_{|I}, P_{\text{out}})$. By the robust soundness property of V_{out} with probability at least $(1 - s_{\text{out}})$ over R_{out}, the string $(\Pi_{|I_{\text{out}}}, P_{\text{out}})$ is at least ρ_{out}-far from satisfying D_{out}. Therefore, with probability at least $(1 - s_{\text{out}})$, the inner verifier V_{in} gets $(D_{\text{out}}, (\Pi_{|I_{\text{out}}}, P_{\text{out}}))$ instance of circuit value problem where $(\Pi_{|I_{\text{out}}}, P)$ is ρ_{out}-far[19] from satisfying D_{out}. Since, the proximity parameter δ_{in} of V_{in} satisfies

---

[19]The guarantee on the input to the circuit is strong than just saying that it is ρ_{out}-far from satisfying D_{out}. This is because the inner verifier always gets the *honest* parities $C_i(R_{\text{out}})$ when needed, unlike the proof queries which can be arbitrary in

Hardwired: Outer verifier V_{out} and inner verifier V_{in}.

Input: Explicit input x, outer proof Π and inner proofs $\{\pi_{R_{\text{out}}} | R_{\text{out}} \in \{0,1\}^{r_{\text{out}}}\}$.

1. Obtain outer verifier circuit D_{out} from $V_{\text{out}}(x)$.[a]

2. Sample $R_{\text{out}} \in \{0,1\}^{r_{\text{out}}}$.

3. Run $V_{\text{out}}(x; R_{\text{out}})$ to obtain $I_{\text{out}} = (i_1, i_2, \ldots, i_{q_{\text{out}}})$ and the parity-checks $P_{\text{out}} = (C_1, \ldots, C_{p_{\text{out}}})$.

4. Sample $R_{\text{in}} \in \{0,1\}^{r_{\text{in}}}$.

5. Run $V_{\text{in}}(D_{\text{out}}, R_{\text{in}})$ to obtain $I_{\text{in}} = ((b_1, j_1), \ldots, (b_{q_{\text{in}}}, j_{q_{\text{in}}}))$ and D_{in}.

6. Initialize $I_{\text{comp}}, P_{\text{comp}} \coloneqq \emptyset$, and $D_{\text{comp}} \coloneqq D_{\text{in}}$

7. For each $k \in [q_{\text{in}}]$, determine the queries of the composite verifier:

   (a) If $b_k = 0$ and $j_k \leq i_{q_{\text{out}}}$, let $\hat{i}_k$ be the location of $\Pi(i_{j_k})$. Append $\hat{i}_k$ to I_{comp}.

   (b) If $b_k = 0$ and $j_k > i_{q_{\text{out}}}$. Append $C_{j_k - q_{\text{out}}}$ to P_{comp}.

   (c) If $b_k = 1$, let $\hat{i}_k$ be the location of $\pi_{R_{\text{out}}}(j_k)$. Append $\hat{i}_k$ to I_{comp}.

Output $(I_{\text{comp}}, P_{\text{comp}}, D_{\text{comp}})$.

---

[a]We use 0-ROP of the outer verifier to obtain the circuit before sampling the outer randomness.

Figure 3: The composite verifier V_{comp} of Fig. 2, adapted to preserve ROP.

$\delta_{\text{in}} \leq \rho_{\text{out}}$, V_{in} rejects any such proofs with probability at least $1 - s_{\text{in}}$. Therefore, the composite verifier rejects the proof with probability at least $(1 - s_{\text{out}})(1 - s_{\text{in}})$ and hence the soundness error is at most $1 - (1 - s_{\text{out}})(1 - s_{\text{in}}) \leq 1 - s_{\text{out}} - s_{\text{in}}$.

RNL   To show RNL of V_{comp}, we show that there is a partition of the composite randomness, as well as a row neighbor-listing agent $\hat{A}_{\text{row}}$ and column neighbor-listing agent $\hat{A}_{\text{col}}$.

Let the RNL partition of the outer randomness R_{out} be given by

$$R = (R_{\text{row}}, R_{\text{col}}, R_{\text{shared.row}}, R_{\text{shared.col}}) \in \{0,1\}^{r_{\text{out}}}.$$

The partition of the composite randomness $\hat{R}$ (which consists of the randomness of both outer and inner verifier) is rather simple: the composite row and column parts are the same as the outer's, and the shared part is formed of the outer's shared part as well as the entire inner randomness. Formally, composite randomness $\hat{R}$ is partitioned into

$$\hat{R}_{\text{row}} \coloneqq R_{\text{row}}$$

$$\hat{R}_{\text{col}} \coloneqq R_{\text{col}}$$

$$\hat{R}_{\text{shared.row}} \coloneqq (R_{\text{shared.row}}, R_{\text{in}})$$

$$\hat{R}_{\text{shared.col}} \coloneqq (R_{\text{shared.col}}, R_{\text{in}}).$$

As before, some bits of shared randomness (in this case, R_{in}) appear in both $R_{\text{shared.row}}$ and $R_{\text{shared.col}}$. Again, we can precisely meet the requirement in Definition 2.9 by further partitioning the inner randomness into two parts. Then, indeed we have

$$\hat{\tau} \coloneqq \frac{|R_{\text{in}}| + |R_{\text{shared.row}}| + |R_{\text{shared.col}}|}{|R_{\text{in}}| + |R|} = \frac{r_{\text{in}} + \tau \cdot r_{\text{out}}}{r_{\text{in}} + r_{\text{out}}}.$$

---

soundness case (See Item 7b in Figure 3). However, we do not need this structure in proving the lemma.

We construct the row and column agents of V_{comp}, denoted $\hat{A}_{\text{row}}$ and $\hat{A}_{\text{col}}$ (respectively). Fix a configuration $(\hat{R}, k) := (R_{\text{out}}, R_{\text{in}}, k)$ where the outer randomness is R_{out} and the inner randomness is R_{in}. We describe only the row agent $\hat{A}_{\text{row}}$, as the column agent $\hat{A}_{\text{col}}$ is analogous. The outcome depends on whether the query was issued to the outer proof or to one of the inner proofs. (In the notation of Fig. 3, this is determined by the value of b_k.)

Case 1 The k-th query is to an inner proof (i.e., $b_k = 1$).

Observe that if a configuration $(R'_{\text{out}}, R'_{\text{in}}, k')$ $(^{20})$ results in a query to the same location, then it must have the same outer randomness as $\left(\hat{R}, k\right)$; that is, $R'_{\text{out}} = R_{\text{out}}$. If indeed $R_{\text{out}} = R'_{\text{out}}$, checking whether the location queried by the two configurations is the same depends only on the inner randomness and the query index, and these are known given the composite shared randomness. In such a case, $\hat{A}_{\text{row}}$ lists all neighboring configurations by finding all possible (R'_{in}, k') that lead to the same location via "brute-force" enumeration. Thus, in this case, on input $(R_{\text{row}}, R_{\text{shared}}, R_{\text{in}})$, the row agent runs as follows:

1. Initialize a list $\hat{L}_{\text{row}}$.
2. For each $R'_{\text{in}} \in \{0,1\}^{r_{\text{in}}}$ and $k' \in [q_{\text{in}}]$ such that $b_{k'} = 1$:
   (a) Check if the inner configuration (R'_{in}, k') results in the same query location as the configuration (R_{in}, k). If so, add $(R_{\text{row}}, R_{\text{shared.row}}, R'_{\text{in}}, k')$ to the list $\hat{L}_{\text{row}}$.
3. Output $\hat{L}_{\text{row}}$.

Case 2 The k-th query is to the outer proof (i.e., $b_k = 0$).

Let $(R'_{\text{out}}, R'_{\text{in}}, k')$ be a configuration that potentially leads to read the same proof location as $(R_{\text{out}}, R_{\text{in}}, k)$. Denote by $(b'_{k'}, j'_{k'})$ the k'-query of the inner verifier on $(D_{\text{out}}, R'_{\text{in}})$.

In this case, we have that $(R_{\text{out}}, R_{\text{in}}, k)$ and $(R'_{\text{out}}, R'_{\text{in}}, k')$ are neighboring configurations if and only if $b'_{k'} = 0$ and the underlying configurations to V_{out}, namely (R_{out}, j_k) and $(R'_{\text{out}}, j'_{k'})$, are neighboring configurations with respect to V_{out}.

Thus, we start by listing all neighboring configurations of (R_{out}, j_k) with respect to V_{out} in a rectangular and synchronized fashion. Then, we extend each such neighbor (R'_{out}, j') to a list of all configurations $(R'_{\text{out}}, R'_{\text{in}}, k')$ to V_{comp} that read the same location in the outer proof.

Thus, in this case, on input $(R_{\text{row}}, R_{\text{shared}}, R_{\text{in}})$, the row agent runs as follows:

1. Initialize a list $\hat{L}_{\text{row}}$.
2. Compute j_k based on D_{out} and R_{in}.
3. Invoke the outer row agent $A_{\text{row}}(R_{\text{row}}, R_{\text{shared}}, j_k)$ to obtain a list L_{row}.
4. For each $(R'_{\text{row}}, R'_{\text{shared.row}}, j')$ in L_{row}:
   (a) For each $R'_{\text{in}} \in \{0,1\}^{r_{\text{in}}}$ and $k' \in [q_{\text{in}}]$:
      i. Emulate V_{in} on input D_{out} and randomness R'_{in}. If the k'-th query is to location j' in the input oracle, i.e., $b_{k'} = 0$ and $j_{k'} = j'$, add $(R'_{\text{row}}, R'_{\text{shared.row}}, R'_{\text{in}}, k')$ to $\hat{L}_{\text{row}}$.
5. Output $\hat{L}_{\text{row}}$.

It is simple to verify that these lists are synchronized and canonical, and that the zip $\hat{L}$ of the two lists $\hat{L}_{\text{row}}$ and $\hat{L}_{\text{col}}$ lists all neighboring configurations of $(\hat{R}, k)$.

We show that $\hat{A}_{\text{row}}$ knows the index of the full given configuration $(\hat{R}, k)$ in $\hat{L}_{\text{row}}$ (an analogous statement holds for $\hat{A}_{\text{col}}$). In case 2, observe that running A_{row} on $(R_{\text{row}}, R_{\text{shared}}, j_k)$ gives the unique index of the entry corresponding to (R_{out}, j_k) in L_{row}. Furthermore, from the shared randomness, $\hat{A}_{\text{row}}$ knows (R_{in}, k). Together this identifies uniquely the list entry in $\hat{L}_{\text{row}}$ that corresponds to (R, k). In case 1, this is even simpler, since from the shared randomness $\hat{A}_{\text{row}}$ knows (R_{in}, k) and can thus identify the unique corresponding list entry in $\hat{L}_{\text{row}}$.

---

[20] We stress that R'_{out} and, R'_{in} are the outer and inner randomness of the configuration.

The running time of both agents is dominated by the double enumeration in [Item 4](#) of Case 2, which is dominated by

$$t_{\mathrm{RNL}} \cdot 2^{r_{\mathrm{in}}} \cdot q_{\mathrm{in}} \cdot t_{\mathrm{in}}. \tag{4}$$

ROP   We prove that the composite verifier has $\hat{\tau}$-ROP, where the aware and shared part of the randomness are the same. Since the outer verifier has 0-ROP, then the description of D_{out} is indeed independent of the randomness. Also, the set of queries $I_{\mathrm{in}} = ((b_1, j_1), \ldots, (b_{q_{\mathrm{in}}}, j_{q_{\mathrm{in}}}))$ and D_{in} depend only on R_{in}, which is a part of the aware randomness. Thus, once the aware randomness is fixed, the query corresponding to (b_c, j_c) where $b_c = 0$ and $j_c > i_{q_{\mathrm{out}}}$, is a fixed parity of the outer randomness R_{out}. Since the aware part of randomness contains the shared outer randomness, the j_c-th input to D_{in} is a fixed affine (rather than linear) function of R_{obliv}.

This shows that the predicate of the composite verifier depends only on the aware randomness, which is the same as the shared randomness. Additionally, the randomness parity checks fed into the predicate are determined by the aware part of the randomness. $\qquad\square$

8   Obtaining the final construct

In this section, we obtain our final *short*, *efficient* and *constant-query* PCP for languages in $\mathsf{NTIME}(T(n))$ that is smooth, rectangular and has ROP.

We will use the randomness-efficient, constant-query PCPP of Mie in the inner level of our composition.

Theorem 8.1 ([Mie09])**.** *Suppose L is a pair language in $\mathsf{NTIME}(T(n))$ for some non-decreasing function $T(n)$. Then, for every constant $0 < s, \delta < 1$, L has a PCPP verifier with the following parameters:*

- *randomness complexity $r_{\mathrm{in}}(n) \coloneqq \log T(n) + O(\log \log T(n))$,*

- *query complexity $q_{\mathrm{in}}(n) \coloneqq O_{s,\delta}(1)$,*

- *verification time $t_{\mathrm{in}}(n) \coloneqq \mathsf{poly}(n, \log K, \log T(n + K))$.*

- *soundness error s and proximity parameter δ.*

Here K is the length of the implicit input y.

We can now prove the following main theorem.

Theorem 8.2. *Let L be a language in $\mathsf{NTIME}(T(n))$ for some non-decreasing function $T \colon \mathbb{N} \to \mathbb{N}$. There exists a universal constant c such that for all odd integers $m \in \mathbb{N}$, and any constant $s \in (0, \frac{1}{2})$ satisfying $T(n)^{1/m} \geq m^{cm}/s^6$, L has a PCP verifier with the following parameters:*

- *Alphabet $\{0, 1\}$.*

- *Randomness complexity $r(n) = \log T(n) + O(m \log \log T(n)) + O(\log n)$.*

- *Soundness error s.*

- *Decision, query and parity-check complexities all $O_s(1)$.*

- *Verifier runtime $t(n) = \mathsf{poly}(n, T(n)^{1/m})$.*

- *The verifier has τ-ROP and is τ-rectangular, where*

$$\tau \cdot r(n) = \frac{6}{m} \log T(n) + O(m \log \log T(n)) + O(\log n).$$

Furthermore, the shared and the aware parts of the randomness are the same.

Proof. We start with the robust PCP verifier over the Boolean alphabet for L given by [Theorem 5.1](#), with the following complexities:

- Randomness complexity $r_{\mathrm{out}}(n) = (1 - \frac{1}{m}) \log T(n) + O(m \log \log T(n)) + O(\log(1/s))$.

- Query and Decision complexity $q_{\mathrm{out}}(n) = d_{\mathrm{out}}(n) = T(n)^{1/m} \cdot \mathsf{poly}(\log T(n), 1/s)$.

- Verifier running time $t_{\mathrm{out}}(n) = q_{\mathrm{out}}(n) \cdot \mathsf{poly}(n, \log T(n))$.

- τ_{out}-RNL, with $\tau_{\mathrm{out}} \cdot r_{\mathrm{out}}(n) = \frac{4}{m} \log T(n) + O(m \log \log T(n)) + O(\log(1/s))$, and listing-agents runtime $t_{\mathrm{RNL,out}}(n) = \mathsf{poly}(\log T(n))$.

- Robust soundness error $s/3$ with robustness parameter $\rho = \Theta(s)$.

We next add the τ_{out}-ROP to this PCP verifier using Lemma 6.2. This step increases the decision complexity and verifier's running time to $\widetilde{O}(t_{\mathrm{out}}(n))$ and adds the parity-check complexity $q_{\mathrm{out}}(n)$. It also reduces the robustness parameter by a constant factor to $\Omega(\rho)$.

Next, we wish to compose this (outer) robust PCP with the inner PCPP of Theorem 8.1. Since the decision complexity of the outer verifier is $\widetilde{O}(t_{\mathrm{out}}(n))$ the inner PCPP ought to verify the pair-language $\mathsf{CVP} \in \mathsf{NTIME}(\widetilde{O}(t_{\mathrm{out}}(n)))$. Therefore, we compose this robust PCP with the PCPP verifier of Mie given in Theorem 8.1, for the pair language $\mathsf{CVP} \in \mathsf{NTIME}(\widetilde{O}(t_{\mathrm{out}}(n)))$, with soundness error $s/3$ and proximity parameter which is greater than the robustness parameter $\Omega(\rho)$ of the outer verifier. The PCPP has query complexity $O_s(1)$, randomness complexity $r_{\mathrm{in}}(\widetilde{O}(t_{\mathrm{out}}(n))) = \frac{1}{m} \log T(n) + O(\log \log T(n)) + O(\log n) + O(\log(1/s))$ and verification time $t_{\mathrm{in}}(\widetilde{O}(t_{\mathrm{out}}(n))) = \mathsf{poly}(t_{\mathrm{out}}(n)) = \mathsf{poly}(n, T(n)^{1/m}, 1/s)$.

By Theorem 7.3 and Lemma 7.4, the composite PCP verifier has soundness error $2s/3$, and query and parity-check complexities both $O_s(1)$. The randomness complexity of the composite verifier, denoted by $r(n)$, is $r_{\mathrm{out}}(n) + r_{\mathrm{in}}(\widetilde{O}(t_{\mathrm{out}}(n)))$. The running time of the composite verifier, denoted $t_{\mathrm{comp}}(n)$, is $\widetilde{O}(t_{\mathrm{out}}(n)) + t_{\mathrm{in}}(\widetilde{O}(t_{\mathrm{out}}(n))) = \mathsf{poly}(t_{\mathrm{out}}(n)) = \mathsf{poly}(n, T(n)^{1/m})$.

The composite verifier has $\hat{\tau}$-RNL and $\hat{\tau}$-ROP where

$$\hat{\tau} = \frac{r_{\mathrm{in}}(\widetilde{O}(t_{\mathrm{out}}(n))) + \tau_{\mathrm{out}} \cdot r_{\mathrm{out}}(n)}{r_{\mathrm{in}}(\widetilde{O}(t_{\mathrm{out}}(n))) + r_{\mathrm{out}}(n)} \leq \tau.$$

Furthermore, the shared and the aware parts of the randomness are indeed the same. The running time of the RNL agents, denoted $t_{\mathrm{RNL}}(n)$, is at most

$$O(t_{\mathrm{RNL,out}}(n) \cdot 2^{r_{\mathrm{in}}(\widetilde{O}(t_{\mathrm{out}}(n)))} \cdot t_{\mathrm{in}}(\widetilde{O}(t_{\mathrm{out}}(n))) \leq \mathsf{poly}(n, T(n)^{1/m}) .$$

Finally, we use Theorem 4.1 with $\mu = s/3$ to make the composite verifier smooth and rectangular. This step increases the soundness error by $s/3$ and thus the overall soundness error is s as required. The PCP verifier becomes smooth and is τ-rectangular and has τ-ROP, with the shared and the aware parts of the randomness still the same. This conversion keeps the randomness and parity-check complexities the same. The decision and query complexities remain the same up to constants. Finally, the running time of the verifier is $t(n) = q \cdot \mathsf{poly}(t_{\mathrm{RNL}}(n)) + t_{\mathrm{comp}}(n) = \mathsf{poly}(n, T(n)^{1/m})$, as asserted. $\qquad\square$

References

[AB09]   SANJEEV ARORA and BOAZ BARAK. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. 3

[AC19]   JOSH ALMAN and LIJIE CHEN. *Efficient construction of rigid matrices using an NP oracle*. In *Proc. 60th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 1034–1055. 2019. 2, 7, 12

[AGHP92]   NOGA ALON, ODED GOLDREICH, JOHAN HÅSTAD, and RENÉ PERALTA. *Simple constructions of almost $k-$wise independent random variables*. Random Structures Algorithms, 3(3):289–304, Fall 1992. (Preliminary version in *31st FOCS*, 1990). 20, 34

[ALMSS98] SANJEEV ARORA, CARSTEN LUND, RAJEEV MOTWANI, MADHU SUDAN, and MARIO SZEGEDY. *Proof verification and the hardness of approximation problems*. J. ACM, 45(3):501–555, May 1998. (Preliminary version in *33rd FOCS*, 1992). eccc:1998/TR98-008. 2

[AS98] SANJEEV ARORA and SHMUEL SAFRA. *Probabilistic checking of proofs: A new characterization of NP*. J. ACM, 45(1):70–122, January 1998. (Preliminary version in *33rd FOCS*, 1992). 2, 6

[AW17] JOSH ALMAN and R. RYAN WILLIAMS. *Probabilistic rank and matrix rigidity*. In *Proc. 49th ACM Symp. on Theory of Computing (STOC)*, pages 641–652. 2017. `arXiv:1611.05558`. 2

[BFLS91] LÁSZLÓ BABAI, LANCE FORTNOW, LEONID A. LEVIN, and MARIO SZEGEDY. *Checking computations in polylogarithmic time*. In *Proc. 23rd ACM Symp. on Theory of Computing (STOC)*, pages 21–31. 1991. 3

[BGHSV05] ELI BEN-SASSON, ODED GOLDREICH, PRAHLADH HARSHA, MADHU SUDAN, and SALIL VADHAN. *Short PCPs verifiable in polylogarithmic time*. In *Proc. 20th IEEE Conf. on Comput. Complexity*, pages 120–134. 2005. Full version available at `http://www.tcs.tifr.res.in/~prahladh/papers/BGHSV2/BGHSV2005.pdf`. 3, 6, 7, 9, 17, 18

[BGHSV06] ———. *Robust PCPs of proximity, shorter PCPs and applications to coding*. SIAM J. Comput., 36(4):889–974, 2006. (Preliminary version in *36th STOC*, 2004). `eccc:2004/TR04-021`. 3, 6, 7, 9, 17, 18, 19, 21, 23, 24, 25, 34

[BLR93] MANUEL BLUM, MICHAEL LUBY, and RONITT RUBINFELD. *Self-testing/correcting with applications to numerical problems*. J. Comput. Syst. Sci., 47(3):549–595, December 1993. (Preliminary version in *22nd STOC*, 1990). 6

[BS08] ELI BEN-SASSON and MADHU SUDAN. *Short PCPs with polylog query complexity*. SIAM J. Comput., 38(2):551–607, 2008. (Preliminary version in *37th STOC*, 2005). `eccc:2004/TR04-060`. 3

[BSVW03] ELI BEN-SASSON, MADHU SUDAN, SALIL VADHAN, and AVI WIGDERSON. *Randomness-efficient low degree tests and short PCPs via epsilon-biased sets*. In *Proc. 35th ACM Symp. on Theory of Computing (STOC)*, pages 612–621. 2003. 3

[BV14] ELI BEN-SASSON and EMANUELE VIOLA. *Short PCPs with projection queries*. In JAVIER ESPARZA, PIERRE FRAIGNIAUD, THORE HUSFELDT, and ELIAS KOUTSOUPIAS, eds., *Proc. 42nd International Colloq. of Automata, Languages and Programming (ICALP), Part I*, volume 8572 of *LNCS*, pages 163–173. Springer, 2014. `eccc:2014/TR14-017`. 3

[Coo88] STEPHEN A. COOK. *Short propositional formulas represent nondeterministic computations*. Inform. Process. Lett., 26(5):269–270, 11 January 1988. 23

[CW16] TIMOTHY M. CHAN and RYAN WILLIAMS. *Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky*. In ROBERT KRAUTHGAMER, ed., *Proc. 27th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1246–1255. 2016. 3, 4, 12

[DE19] ZEEV DVIR and BENJAMIN L. EDELMAN. *Matrix rigidity and the Croot-Lev-Pach lemma*. Theory Comput., 15(8):1–7, 2019. `arXiv:1708.01646`. 2

[DH13] IRIT DINUR and PRAHLADH HARSHA. *Composition of low-error 2-query PCPs using decodable PCPs*. SIAM J. Comput., 42(6):2452–2486, 2013. (Preliminary version in *51st FOCS*, 2009). `eccc:2009/TR09-042`. 15, 17

[Din07] IRIT DINUR. *The PCP theorem by gap amplification*. J. ACM, 54(3):12, 2007. (Preliminary version in *38th STOC*, 2006). `eccc:2005/TR05-046`. 3

[DL19] ZEEV DVIR and ALLEN LIU. *Fourier and circulant matrices are not rigid*. In *Proc. 34th Comput. Complexity Conf.*, volume 137 of *LIPIcs*, pages 17:1–17:23. Schloss Dagstuhl, 2019. `arXiv:1902.07334`, `eccc:2019/TR19-129`. 2

[DR06] IRIT DINUR and OMER REINGOLD. *Assignment testers: Towards a combinatorial proof of the PCP Theorem*. SIAM J. Comput., 36:975–1024, 2006. (Preliminary version in *45th FOCS*, 2004). 6

[For65] GEORGE DAVID FORNEY, JR. *Concatenated Codes*. Ph.D. thesis, Massachusetts Institute of Technology, 1965. 21

[Fri93] JOEL FRIEDMAN. *A note on matrix rigidity*. Combinatorica, 13(2):235–239, 1993. 2

[Gol11] ODED GOLDREICH. *A sample of samplers: A computational perspective on sampling*. In ODED GOLDREICH, ed., *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, volume 6650 of *LNCS*, pages 302–332. Springer, 2011. `eccc:1997/TR97-020`. 15

[GS06] ODED GOLDREICH and MADHU SUDAN. *Locally testable codes and PCPs of almost linear length*. J. ACM, 53(4):558–655, 2006. (Preliminary version in *43rd FOCS*, 2002). `eccc:2002/TR02-050`. 9

[GT18]   Oded Goldreich and Avishay Tal. *Matrix rigidity of random Toeplitz matrices*. Comput. Complexity, 27(2):305–350, 2018. Preliminary version in *48th STOC*, 2002). `eccc:2015/TR15-079`. 2

[IKW02]   Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. *In search of an easy witness: exponential time vs. probabilistic polynomial time*. J. Comput. Syst. Sci., 65(4):672–694, 2002. (Preliminary version in *16th Computational Complexity Conference*, 2002). 3

[KT00]   Jonathan Katz and Luca Trevisan. *On the efficiency of local decoding procedures for error-correcting codes*. In *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pages 80–86. 2000. 9

[Lok09]   Satyanarayana V. Lokam. *Complexity lower bounds using linear algebra*. Foundations and Trends in Theoretical Computer Science, 4(1-2):1–155, 2009. 2

[Mie09]   Thilo Mie. *Short PCPPs verifiable in polylogarithmic time with o(1) queries*. Ann. Math. Artif. Intell., 56(3-4):313–338, 2009. 3, 7, 28

[MR08]   Dana Moshkovitz and Ran Raz. *Sub-constant error low degree test of almost-linear size*. SIAM J. Comput., 38(1):140–180, 2008. (Preliminary version in *38th STOC*, 2006). `eccc:2005/TR05-086`. 3

[NN93]   Joseph Naor and Moni Naor. *Small-bias probability spaces: Efficient constructions and applications*. SIAM J. Comput., 22(4):838–856, August 1993. (Preliminary version in *22nd STOC*, 1990). 20

[Par20]   Orr Paradise. *Smooth and strong PCPs*. In Thomas Vidick, ed., *Proc. 11th Innovations in Theor. Comput. Sci. (ITCS)*, volume 151 of *LIPIcs*, pages 2:1–2:41. Schloss Dagstuhl, 2020. `eccc:2019/TR19-023`. 3, 9

[PF79]   Nicholas Pippenger and Michael J. Fischer. *Relations among complexity measures*. J. ACM, 26(2):361–381, April 1979. 18

[Raz89]   Alexander A. Razborov. *On rigid matrices (in Russian)*, 1989. (manuscript). 2

[Spi96]   Daniel A. Spielman. *Linear-time encodable and decodable error-correcting codes*. IEEE Trans. Inform. Theory, 42(6):1723–1732, November 1996. (Preliminary version in *27th STOC*, 1995). 21

[SSS97]   Mohammad Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. *A remark on matrix rigidity*. Inform. Process. Lett., 64(6):283–285, 1997. 2

[Tre13]   Luca Trevisan. *Inapproximability of combinatorial optimization problems*. In Vangelis Th. Paschos, ed., *Paradigms of Combinatorial Optimization*, chapter 13, pages 381–434. Wiley Publishers, 2013. `arXiv:cs.CC/0409043`, `eccc:2004/TR04-065`. 3

[Val77]   Leslie G. Valiant. *Graph-theoretic arguments in low-level complexity*. In Jozef Gruska, ed., *Proc. 6th Symposium of Mathematical Foundations of Computer Science (MFCS)*, volume 53 of *LNCS*, pages 162–176. Springer, 1977. 2

[Vio20]   Emanuele Viola. *New lower bounds for probabilistic degree and AC^0 with parity gates*, 2020. (manuscript). `eccc:2020/TR20-015`. 7

[Wun12]   Henning Wunderlich. *On a Theorem of Razborov*. Comput. Complexity, 21(3):431–477, 2012. `eccc:2010/TR10-086`. 2

[Zák83]   Stanislav Zák. *A Turing machine time hierarchy*. Theoret. Comput. Sci., 26:327–333, 1983. 3, 4, 14

A   Soundness and smoothness of Algorithm 4.4

With rectangularity out of the way, we can describe the run of the new verifier given proof π_{new} a more succinct way:

Algorithm A.1 (Algorithm 4.4, simplified)**.** Given input x and proof π_{new}, the new verifier V_{new} runs as follows:

1. Sample $R \in \{0,1\}^r$.

2. For each $k \in [q]$, query π_{new} for (R,k) as well as its sampler-neighbors.

3. Feed the $q \cdot \Delta$ bits queried from the proof π_{new} to a circuit that first checks consistency between every sampler-neighborhood. That is, it check that in each of the q blocks of Δ bits, all the Δ bits are equal. If an inconsistency is spotted the circuit immediately rejects. Otherwise, feed the first bit in every block to the decision circuit of the original verifier V_{old} (along with the p parity-checks on the randomness) and output its answer.

Indeed, Algorithm 4.4 and Algorithm A.1 describe the same verifier precisely due to RNL: for any random coin sequence R and query index k, Item 2 of Algorithm 4.4 indeed queries all neighbors of (R, k) in the sampler (and then checks their consistency). We now show that this verifier is sound and smooth.

A.1 Soundness

Let $\alpha := \mu/2q$. Recall that V_{new} denotes the new smooth verifier, and V_{old} denotes the original verifier. Fix an input $x \notin L$ and an alleged proof π_{new} for V_{new}. We will show that V_{new} rejects x and π_{new} with probability at least $1 - s - \mu$.

We let $i^{(k)}(R)$ denote the location of the k-th query of V_{old} when sampling random coins R. Recall that locations in π_{new} are indexed by full configurations (R, k), that can be partitioned into m disjoint α-samplers, where the j-th sampler connects all configurations (R, k) such that $i^{(k)}(R) = j$. We derive a proof π_{old} for the original verifier V_{old} by assigning $\pi_{\text{old}}(j)$ the majority value of π_{new} on the j-th sampler. Formally,

$$\pi_{\text{old}}(j) := \mathsf{Maj}_{R,k}\Big\{\pi_{\text{new}}(R, k) \mid i^{(k)}(R) = j\Big\}.$$

We say that $j \in [m]$ has consistency $1 - 2\alpha$ if $(1 - 2\alpha)$-fraction of the configurations (R, k) pointing to it satisfy $\pi_{\text{new}}(R, k) = \pi_{\text{old}}(j)$.

Two events in the verifier's randomness R will be of particular interest:

- H: for all $k \in [q]$, $i^{(k)}(R)$ has consistency at least $1 - 2\alpha$.

- M: for all $k \in [q]$, $\pi_{\text{new}}(R, k) = \pi_{\text{old}}\big(i^{(k)}(R)\big)$.

Let $\delta = \Pr\big[\overline{H}\big]$. We show the following two claims.

Claim A.2.
$$\Pr\big[V_{\text{new}}{}^{\pi_{\text{new}}} = 0 | \overline{H}\big] \geq 1 - q\alpha.$$

Claim A.3.
$$\Pr\big[V_{\text{new}}{}^{\pi_{\text{new}}} = 0 | H\big] \geq 1 - \frac{s}{1 - \delta} - 2q\alpha.$$

These two claims will give the required result, because

$$\begin{aligned}
\Pr\big[V_{\text{new}}{}^{\pi_{\text{new}}} = 0\big] &= \Pr\big[\overline{H}\big] \cdot \Pr\big[V_{\text{new}}{}^{\pi_{\text{new}}} = 0 | \overline{H}\big] + \Pr\big[H\big] \cdot \Pr\big[V_{\text{new}}{}^{\pi_{\text{new}}} = 0 | H\big] \\
&\geq \delta(1 - q\alpha) + (1 - \delta)\left(1 - \frac{s}{1 - \delta} - 2q\alpha\right) \\
&= \delta - \delta q\alpha + (1 - \delta) - s - (1 - \delta)2q\alpha \\
&\geq 1 - s - 2q\alpha = 1 - s - \mu.
\end{aligned}$$

Now let's prove the claims.

Proof of Claim A.2. Consider a sampler with consistency at most $1 - 2\alpha$. The average value[21] of π_{new} on this sampler is between 2α and $1 - 2\alpha$. Since this is a α-sampler, it holds that for all but α-fraction of configurations (named error configurations), the average value of π_{new} on the sampler-neighborhood of each configuration is between α and $1 - \alpha$. In particular, π_{new} assigns inconsistent values to the sampler-neighbors of each non-error configuration.

Denoting the set of all error configurations by E, we see that sampling an $R \in \overline{H}$ such that $(R, k) \notin E$ for all k leads the V_{new} to reject in the consistency check, Item 2 of Algorithm A.1. We upper bound the probability of the complement event,

$$\Pr_R\big[\exists k \in [q] \quad (R, k) \in E | \overline{H}\big] \leq q \cdot \Pr_{R,k}\big[(R, k) \in E | \overline{H}\big] \leq q \cdot \alpha$$

and the claim follows. □

---

[21]Recall that π_{new} is over $\{0, 1\}$ and we interpret these as real numbers.

Proof of Claim A.3. To prove this claim, we will show the following lower bounds:

$$\Pr\left[M|H\right] \geq 1 - 2q\alpha, \tag{5}$$

$$\Pr\left[V_{\text{old}}^{\pi_{\text{old}}} = 0|H\right] \geq 1 - \frac{s}{1-\delta}. \tag{6}$$

Once those are in place, the claim follows by noticing that if M occurs, then either the new verifier V_{new} rejects for inconsistency or its queries to π_{new} give the same answers as the answers by V_{old} to π_{old}. Thus, conditioned on M, if V_{old} rejects π_{old} on randomness R, then so does V_{new} on π_{new}. This gives

$$\begin{aligned}
\Pr\left[V_{\text{new}}^{\pi_{\text{new}}} = 0|H\right] &\geq \Pr\left[M|H\right] \cdot \Pr\left[V_{\text{new}}^{\pi_{\text{new}}} = 0|H \cap M\right] \\
&\geq \Pr\left[M|H\right] \cdot \Pr\left[V_{\text{old}}^{\pi_{\text{old}}} = 0|H \cap M\right] \\
&= \Pr\left[M \cap V_{\text{old}}^{\pi_{\text{old}}} = 0|H\right] \\
&\geq 1 - \frac{s}{1-\delta} - 2q\alpha.
\end{aligned}$$

To see Eq. (5), let $H_{k'}$ denote the event that $i(R, k')$ has consistency at least $1 - 2\alpha$ (thus, $H = \bigcap_{k'} H_{k'}$). Then,

$$\begin{aligned}
\Pr_R\left[\overline{M} \mid H\right] &= \Pr_R\left[\exists k \quad \pi_{\text{new}}(R, k) \neq \pi_{\text{old}}\left(i^{(k)}(R)\right) \,\Big|\, \bigcap_{k'} H_{k'}\right] \\
&\leq \sum_{k=1}^{q} \Pr_R\left[\pi_{\text{new}}(R, k) \neq \pi_{\text{old}}\left(i^{(k)}(R)\right) \,\Big|\, \bigcap_{k'} H_{k'}\right] \\
&\leq \sum_{k=1}^{q} \Pr_R\left[\pi_{\text{new}}(R, k) \neq \pi_{\text{old}}\left(i^{(k)}(R)\right) \,\Big|\, H_k\right] \\
&\leq q \cdot 2\alpha.
\end{aligned}$$

For Eq. (6), recall that the soundness of the original verifier to gives $\Pr\left[V_{\text{old}}^{\pi_{\text{old}}} = 0\right] \geq 1 - s$. Then, using elementary probability, we have

$$\Pr\left[V_{\text{old}}^{\pi_{\text{old}}} = 0|H\right] \geq \frac{\Pr\left[V_{\text{old}}^{\pi_{\text{old}}} = 0\right] - \Pr\left[\overline{H}\right]}{\Pr\left[H\right]} \geq \frac{1 - s - \delta}{1 - \delta} = 1 - \frac{s}{1-\delta}. \qquad \square$$

A.2 Smoothness

For each $R \in \{0,1\}^r$ and $k \in [q]$, let $\overline{\Gamma}(R, k)$ denote the closed sampler-neighborhood of (R, k), i.e. the union of the sampler-neighborhood of (R, k) and the singleton containing it $\{(R, k)\}$. Recall that the sampler is a regular graph with constant degree $\Delta - 1$, and thus $\Delta = \left|\overline{\Gamma}(R, k)\right|$ for each R, k.

Now recall how the new verifier determines its queries (see Algorithm A.1): sample $R \in \{0,1\}^r$, and for each $k \in [q]$ query all the Δ locations in $\overline{\Gamma}(R, k)$. We must now show why this procedure is equally likely to query each location in π_{new}.

Fix a location (R', k') in the new proof. Notice that

$$\Pr_{\substack{R \in \{0,1\}^r \\ k \in [q]}}\left[(R', k') \in \overline{\Gamma}(R, k)\right] = \frac{\Delta}{2^r \cdot q}, \tag{7}$$

where both R and k are distributed uniformly and random. Recall that the new verifier makes $q \cdot \Delta$ queries to the proof, where the (k, j)-th query, for $k \in [q]$ and $j \in [\Delta]$, is $\overline{\Gamma}(R, k)[j]$. We thus get

$$\Pr_{\substack{R \in \{0,1\}^r \\ k \in [q], j \in [\Delta]}}\left[(R', k') = \overline{\Gamma}(R, k)[j]\right] = \frac{1}{2^r \cdot q}$$

showing that each location (R', k') is equally likely to be queried by the new verifier.

B   Representing an affine function as a low-rank matrix

Claim B.1. *There is a procedure with the following properties:*

- *Input: An integer m, Boolean vectors $u, v \in \mathbb{F}_2^m$ and a bit $b \in \mathbb{F}_2$.*

- *Output: Two matrices, $A \in (\mathbb{F}_2)^{2^m \times 3}$ and $B \in (\mathbb{F}_2)^{3 \times 2^m}$, such that $(A \cdot B)_{x,y} = \langle x, u \rangle + \langle y, v \rangle + b$.*

- *Runtime $\widetilde{O}(2^m)$.*

Proof. We compute A column-by-column.

- The first column is an enumeration of $\langle x, u \rangle$, for all $x \in \mathbb{F}_2^m$.

- The second column is the all-ones vector, denoted by $\vec{1}$.

- The third column is $b\vec{1}$.

Next, we compute B row-by-row.

- The first row is $\vec{1}$.

- The second row is an enumeration of $\langle y, v \rangle$, for all $y \in \mathbb{F}_2^m$.

- The third row is $\vec{1}$

It is easy to verify that $(A \cdot B)_{x,y} \equiv \langle x, u \rangle + \langle y, v \rangle + b$, and that the runtime is $O(2^m \cdot m) = \widetilde{O}(2^m)$. $\square$

C   The tests and queries of [BGHSV06, Section 8.2.1]

We list the tests performed by the verifier of [BGHSV06, Section 8.2.1]. Since we only reason about RNL of this verifier, describing the *query pattern* of each test suffices. The reader is referred to [BGHSV06, Section 8] for a complete description (i.e., query pattern and the decision predicate).

1. Let $\lambda \le \min\{\log n/c, s^3/m^{cm}\}$. Construct set $S_\lambda \subseteq \{1\} \times \mathbb{F}^{m-1}$, a λ-biased set of size at most $O\left(\frac{\log(|\mathbb{F}|^m)}{\lambda}\right)^2$ (see [AGHP92]).

2. Sample a random string R of length $\log(|S_\lambda| \cdot |\mathbb{F}|^{m-1}) = (m-1)\log(|\mathbb{F}|) + O(\log(1/\lambda)) + O(\log\log(|\mathbb{F}|^m))$. Based on this random string, issue queries for the following tests:

   (a) Robust Edge-Consistency Test$^\Pi(R)$: Use random string R to determine a random 1st axis-parallel line in $\mathbb{F}^m$ of the form $\mathcal{L} = \{(t, a_2, \cdots, a_m)\}_{t \in \mathbb{F}}$ for some fixed $a_2, \ldots, a_m \in \mathbb{F}$. Query the oracle Π on all points along the line $\mathcal{L}$ and $S(\mathcal{L})$.

   (b) Robust Zero Propagation Test$^\Pi(R)$: Use random string R to determine a random 1st axis-parallel line in $\mathbb{F}^m$ of the form $\mathcal{L} = \{(t, a_2, \cdots, a_m)\}_{t \in \mathbb{F}}$ for some fixed $a_2, \ldots, a_m \in \mathbb{F}$. Query the oracle Π on all points along the line $\mathcal{L}$ and $S(\mathcal{L})$.

   (c) Robust Identity Test$^\Pi(R)$: Use random string R to determine a random 1st axis-parallel line in $\mathbb{F}^m$ of the form $\mathcal{L} = \{(t, a_2, \cdots, a_m)\}_{t \in \mathbb{F}}$ for some fixed $a_2, \ldots, a_m \in \mathbb{F}$. Query the oracle Π on all points along the line $\mathcal{L}$.

   (d) Robust Low-Degree Test$^\Pi(R)$: Use random string R to determine a random canonical line $\mathcal{L}$ in $\mathbb{F}^m$ using the λ-biased set S_λ. Query the oracle Π on all points along the line $\mathcal{L}$.