Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 182 (2025)

Proving the PCP Theorem with 1.5 proof compositions (or yet another PCP construction)

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science, Rehovot, Israel.

November 17, 2025

Abstract

The original proof of the PCP Theorem composes a Reed-Muller-based PCP with itself, and then composes the resulting PCP with a Hadamard-based PCP [Arora, Lund, Motwani, Sudan and Szegedy (*JACM*, 1998)]. Hence, that proof applies a (general) proof composition result twice. (Dinur's alternative proof consists of logarithmically many gap amplification steps, where each step includes an operation akin to proof composition.)

A recent work of Amireddy, Behera, Srinivasan, Sudan, and Willumsgaard (*ECCC*, TR25-165) presents a new PCP system such that composing the RM-based PCP with the new PCP yields a new proof of the PCP Theorem. Essentially, for every $\alpha > 0$, they present a (direct) PCP of constant query complexity and randomness complexity n^α, where n denotes the length of the input. (In contrast, recall that the Hadamard-based PCP has quadratic randomness complexity.) We note that their construction has other merits, beyond the fact that it does not use composition.

Here we present a different PCP system of constant query complexity and randomness n^α. Essentially, we use the RM-based PCP, but with a constant number of dimensions and a large alphabet (equiv., $O(1/\alpha)$-variate polynomials over a field of size $n^{O(\alpha)}$). We then encode the field elements by the Hadamard code and incorporate a tester akin to the verifier used in the Hadamard-based PCP. Whether or not this counts as composition is debatable (and this is reflected in the current title), but for sure this is a non-generic composition that does not involve a preparatory parallelization step.

Contents

ISSN 1433-8092

1 Introduction

The original proof of the PCP Theorem (by Arora, Lund, Motwani, Sudan and Szegedy [2]) composes a Reed-Muller-based PCP with itself, and then composes the resulting PCP with a Hadamard-based PCP.[1] Hence, that proof applies a (general) proof composition result twice, where these applications require a preparatory parallelization step. The fact that two proof-composition steps are required seems a coincidence that arises from the parameters of the two aforementioned PCP systems. Specifically, the RM-based PCP has polylogarithmic query complexity, whereas the H-based PCP has quadratic randomness complexity.

Dinur's alternative proof of the PCP Theorem [6] consists of logarithmically many gap amplification steps, where each step includes an operation akin to proof composition (but does not require a preparatory parallelization step).

A recent work of Amireddy, Behera, Srinivasan, Sudan, and Willumsgaard presents a new PCP system such that composing the RM-based PCP with the new PCP yields a new proof of the PCP Theorem [1]. Essentially, for every $\alpha > 0$, they present a (direct) PCP of constant query complexity and randomness complexity n^α, where n denotes the length of the input. We note that their construction has other merits, beyond the fact that it does not use composition. In particular, they can derive both the new PCP system and the known RM-based PCP system as special cases of general construction schema.

The latter work solves a previously untouched challenge: *Constructing a simple PCP system, say for 3SAT, of constant query complexity and randomness complexity n^α*. The undefined qualification (i.e, "simple") is aimed to rule out the PCP system that is obtained by composing the RM-based system with the H-system, resulting in a PCP of constant query complexity and polylogarithmic randomness complexity. (More generally, it is meant to rule out the use of a general proof composition technique and especially the preparatory parallelization step, alas these terms are undefined too.)

In this paper we provide an alternative solution to the foregoing challenge. Essentially, we use the RM-based PCP (of [4, 8, 2]), but with a constant number of dimensions and a large alphabet (equiv., $O(1/\alpha)$-variate polynomials over a field of size $n^{O(\alpha)}$). We emulate this RM-based PCP by using Hadamard encoding of the answers that appear in the corresponding proof-oracle. Details follow (albeit they rely on familiarity with the RM-based PCP system).

Recall that the univariate polynomials returned by the sum-check procedure and by the low-degree test are typically represented by their (sequence of) coefficients. We encode each of these sequences by the Hadamard code; that is, the proof-oracle that we use provide these encodings. Recall that the claim that a univariate polynomial evaluates to a claimed value at a given point is captured by a lineaer constraint on the (sequence of) coefficients of the polynomial (since univariate polynomial evaluation is a linear combination of the coefficients of the polynomial). Next, observe that, when using a field of characteristic 2, a linear combination of a sequence of field elements corresponds to linear combinations of the bits in the representations of these field elements. Hence, testing a linear constraint on the (sequence of) coefficients reduces to inspecting few (random) locations in the Hadamard encoding of this sequence. (Indeed, we shall compare random linear combinations of the bits representing the two field elements (to be compared).)

Hence, we can emulate the sum-check and low-degree tests, leaving us with the task of verifying

[1] Actually, proof composition requires the outer system to be robust and the inner system to be a PCP of Proximity (see below). The parallelization step (mentioned next) is meant to provide robustness.

that the product of three field elements equals a fourth one, where each of these field elements is encoded by a Hadamard code. The latter verification is assisted by an auxiliary proof-oracle that provides the Hadamard encoding of all three-way products of bits (in these three field elements). The corresponding tester is analogous to the verifier used in the Hadamard-based PCP (of [2]).

It may be argued that the latter fact means that we actually perform proof composition; we beg to differ, but postpone this rather scholastic discussion to Section 4. For sure, we neither use a general composition result (of the type presented in [3, 2, 5, 7]) nor a preparatory parallelization step (as in, e.g., [2]).

Since the main application of the PCP system presented here is for composing it (as an inner verifier) with an outer verifier (i.e., the robust RM-based verifier of [2]), we actually construct a *PCP for Proximity* (PCPP).[2] Indeed, composing the (robust) RM-based verifier of [2] with our PCPP yields a PCP system that establishes the PCP Theorem.

Prerequisites. The current text assumes familiarity with the notions of a PCP and a PCPP as well as with the constructions of the RM-based PCP of [4, 8, 2] and the H-based PCP of [2]. A high-level exposition of these notions and constructions can be found in [9, Sec. 9.3.1-9.3.2].

Organization. In Section 2 we provide a high-level description of our PCP system. We believe that this high-level description suffices, but provide a more detailed description in Section 3. Since this PCP system incorporates ingredients that were analyzed in the literature (cf., [2]), we see little point in providing a full analysis. In Section 4 we briefly articulate our disagreement with the claim that our PCP system actually performs proof composition.

2 High-level description

The proposed construction is based on the RM-based PCPP, but it uses a different setting of parameters. Specifically, for any constant $m \in \mathbb{N}$, we use $h = |H| = n^{1/m}$ and $|\mathcal{F}| = \text{poly}(|H|)$. (In contrast, the standard construction uses $m = \frac{\log n}{\log \log n}$ and $h = \log n$.) This PCP has logarithmic randomness, but its query complexity is $\Theta(m \cdot h \log n)$. So we shall not use it as is, but rather encode the answers (to its queries) by the Hadamard code and test these encodings instead.[3] However, let us first take a closer look at the RM-based PCPP.

The RM-based PCPP. First, we stress that we refer to the basic RM-based PCPP *without the parallelization step*. Given (a succinct description of) a 3CNF formula $\phi : \{0,1\}^n \to \{0,1\}$ and oracle access to an assignment $A : [n] \to \{0,1\}$ to its variables, the verifier uses several proof-oracles to be detailed below. Associating $[n]$ with H^m, the first proof-oracle is a low-degree extension of $A : H^m \to \{0,1\}$, denoted $\widehat{A} : \mathcal{F}^m \to \mathcal{F}$; that is, $\widehat{A}$ has individual degree $h-1$, and $\widehat{A}(x) = A(x)$ for every $x \in H^m$.

The verifier tests that $\widehat{A}$ indeed extends A and that $\widehat{A}$ has low degree. The first test is straightforward (i.e., by querying both oracles on $O(1/\epsilon)$ random points in H^m), and it will be ignored in the rest of this section. The second test may be performed by picking a random line in $\mathcal{F}^m$,

[2] Indeed, we prefer the formalism of PCPP, as introduced in [5], over the one of *assignment testers*, introduced independently in [7].

[3] Indeed, this is analogous to performing proof composition with the Hadamard-based PCPP, but we achieve the same effect by capitalizing on the specifics of the current case rather than by invoking a general composition theorem.

obtaining the corresponding (degree $m \cdot (h-1) < mh$) univariate polynomial from a corresponding proof oracle, denoted $\Pi_{\texttt{lines}} : (\mathcal{F}^m)^2 \to \mathcal{F}^{mh}$, and comparing the value at a random point on this line to the value obtained from $\widehat{A}$.

Assuming that $\widehat{A}$ is a low-degree extension of A, the verifier tests that $\widehat{A}$ (restricted to H^m) corresponds to a satisfying assignment to the formula ϕ. This is done by checking that a specific expression evaluates to zero, where the expression has the form[4]

$$\sum_{\xi,x,y,z \in H^m} \Phi(\xi,x,y,z) \cdot \widehat{A}(x) \cdot \widehat{A}(y) \cdot \widehat{A}(z), \tag{1}$$

where $\Phi : \mathcal{F}^{4m} \to \mathcal{F}$ is an explicit polynomial of individual degree $h-1$ that depends on ϕ. Indeed, Eq. (1) is verified by the celebrated sum-check procedure (originating in [10]) that utilizes the oracles $\Pi_1, ..., \Pi_{4m}$, where $\Pi_j : \mathcal{F}^{j-1} \to \mathcal{F}^h$, such that $P_{r_1,...,r_{j-1}} \stackrel{\text{def}}{=} \Pi_j(r_1, ..., r_{j-1})$ is a univariate polynomial of degree $h-1$ that satisfies

$$P_{r_1,...,r_{j-1}}(\zeta) = \sum_{\overline{r} \in H^{4m-j}} \Phi(r_1, ..., r_{j-1}, \zeta, \overline{r}) \cdot \widehat{A}(x) \cdot \widehat{A}(y) \cdot \widehat{A}(z), \tag{2}$$

where $(\xi, x, y, z) = (r_1, ..., r_{j-1}, \zeta, \overline{r})$. Starting with ($j = 1$ and) $v_\lambda = 0$, in the j^{th} iteration, the verifier checks that

$$v_{r_1,...,r_{j-1}} = \sum_{r \in H} P_{r_1,...,r_{j-1}}(r), \tag{3}$$

selects $r_j \in \mathcal{F}$ uniformly at random, and defines $v_{r_1,...,r_j} \leftarrow P_{r_1,...,r_{j-1}}(r_j)$. After the last iteration, the verifier checks that $v_{r_1,...,r_{4m}}$ equals

$$\Phi(r_1, ..., r_{4m}) \cdot \widehat{A}(r_{m+1}, ..., r_{2m}) \cdot \widehat{A}(r_{2m+1}, ..., r_{3m}) \cdot \widehat{A}(r_{3m+1}, ..., r_{4m}). \tag{4}$$

Hence, the queries that this verifier performs and the answers it receives are as follows.

1. The $4m$ queries of the sum-check procedure are each answered by a univariate polynomial of degree $h-1$, which we view as an element of $\mathcal{F}^h$.

   Note that Eq. (3) calls for evaluating these polynomials at $h+1$ different points (i.e., the polynomial $\Pi_j(r_1, ..., r_{j-1}) : \mathcal{F} \to \mathcal{F}$ is evaluated at $H \cup \{r_j\}$). Assuming that each of these univariate polynomials (of degree $h-1$) is represented by the h-long sequence of coefficients, its evaluation corresponds to a linear combination of these coefficients. Specifically, if $P_{r_1,...,r_{j-1}} = \Pi_j(r_1, ..., r_{j-1})$ is represented by $(c_0, ..., c_{h-1}) \in \mathcal{F}^h$ (i.e., $P_{r_1,...,r_{j-1}}(\zeta) = \sum_{i=0}^{h-1} c_i \cdot \zeta^i$), then evaluating this polynomial at $r \in \mathcal{F}$ corresponds to computing $\sum_{i=0}^{h-1} r^i \cdot c_i$.

2. The three queries at the end of the sum-check procedure are each answered by an element of $\mathcal{F}$. These answers are provided by $\widehat{A}$.

   In this case, the corresponding test calls for multiplying these three values of $\widehat{A}$ (and comparing the answer to $v_{r_1,...,r_{4m}}/\Phi(r_1, ..., r_{4m})$).[5]

[4] For simplicity, we assume here that ϕ is monotone; this is justified in Section 3. In addition, we represent `true` (resp., `false`) by 0 (resp., by 1), allows us to use $\widehat{A}(x) \cdot \widehat{A}(y) \cdot \widehat{A}(z)$ rather than $(1-\widehat{A}(x)) \cdot (1-\widehat{A}(y)) \cdot (1-\widehat{A}(z))$. As for Φ itself, it is derived analogously to [9, Eq. (9.8)], while assuming that the number of clauses equals the number of variables. In addition, for simplicity, we omit here the sequence of pseudorandom coefficients $(\omega_\xi)_{\xi \in H^m}$ that are generated based on a random $O(\log n)$-bit long seed, which is selected by the verifier.

[5] Recall that $v_{r_1,...,r_{4m}} = P_{r_1,...,r_{4m-1}}(r_{4m})$, and that the verifier can compute $\Phi(r_1, ..., r_{4m})$ by itself.

3. The first query of the low-degree test is answered by a univariate polynomial of degree $m \cdot (h-1)$, which we view as an element of $\mathcal{F}^{mh}$, whereas the second query is answered by an element of $\mathcal{F}$. (These answers are provided by $\Pi_{\mathtt{lines}}$ and $\widehat{A}$, respectively.)

   Analogously to Step 1, the test calls for evaluating the univariate polynomial at some point, and this is done analogously.

While the foregoing verifier makes $O(m)$ queries, the answers to these queries are represented by sequences over $\mathcal{F}$, which we shall view as bit strings (of lengths between $\log_2|\mathcal{F}|$ and $mh \cdot \log_2|\mathcal{F}|$). In contrast, we aim at $O(m)$ queries that are answers by binary values. As stated upfront, we achieve this goal by encoding the answers by (even) longer (binary) codewords that we shall test in a local manner. Specifically, we shall use the Hadamard code (as well as the known codeword test and self-correction procedure for it).

Using Hadamard encoding of the answers. We first address the queries, answers and checks performed in Steps 1 and 3. Recall that these checks consist of $\mathcal{F}$-linear constraints on the answers. Now, suppose that $\mathcal{F}$ has characteristic 2; that is, $\mathcal{F} = \mathrm{GF}(2^t)$. Then, we can write each $\mathcal{F}$-linear constraint (which is checked in Steps 1 and 3) as t linear constraints over $\mathrm{GF}(2)$, which means that we can easily checked them if we have a Hadamard encoding of these answers (and use self-correction). Of course, we should also perform Hadamard-codeword tests. All these can be done by tossing $O(mh\log|\mathcal{F}|) = \widetilde{O}(|H|)$ coins and making a constant number of queries. Details follow.

Formally, rather than using the oracles $\Pi_j : \mathcal{F}^{j-1} \to \mathcal{F}^h$, we shall use the oracles $\Pi'_j : \mathcal{F}^{j-1} \times \mathcal{F}^h \to \{0,1\}$ such that $\Pi'_j(\overline{r}, s)$ equals the inner-product mod 2 of $\Pi_j(\overline{r})$ and s (each viewed as an $\log_2|\mathcal{F}^h|$-bit long string). Analogously, rather than using $\Pi_{\mathtt{lines}} : (\mathcal{F}^m)^2 \to \mathcal{F}^{mh}$ (resp., $\widehat{A} : \mathcal{F}^m \to \mathcal{F}$), we shall use $\Pi'_{\mathtt{lines}} : (\mathcal{F}^m)^2 \times \mathcal{F}^{mh} \to \{0,1\}$ (resp., $\widehat{A}' : \mathcal{F}^m \times \mathcal{F} \to \{0,1\}$) such that $\Pi'_{\mathtt{lines}}((x,y),s)$ equals the inner-product mod 2 of $\Pi_{\mathtt{lines}}(x,y)$ and s, each viewed as an $\log_2|\mathcal{F}^{mh}|$-bit long string (resp., $\widehat{A}'(z,s)$ equals the inner-product mod 2 of $\widehat{A}(z)$ and s, each viewed as an $\log_2|\mathcal{F}|$-bit long string). Hence, rather than checking one $\mathcal{F}$-linear constraint, we need to check $t = \log_2|\mathcal{F}|$ binary constraints. Needless to say, we shall not do that either, but rather check a random linear combination of these t constraints (and do so via self-correction of the relevant Hadamard-encoding, which will be also tested for being a valid codeword).

Specifically, a linear constraint on the bits of $\Pi_j(\overline{r}) \in \mathcal{F}^h$ (viewed as an $h \cdot t$-bit long string) is checked by considering the corresponding position in Π'_j (i.e., if $s \in \{0,1\}^{ht}$ represents this linear constraint, then we consider $\Pi'_j(\overline{r}, s)$, and reconstruct its purported value by self-correction (i.e., use $\Pi'_j(\overline{r}, s') + \Pi'_j(\overline{r}, s - s')$, where s' is uniformly selected in $\{0,1\}^{ht}$)). Likewise, a linear constraint on the bits of $\Pi_{\mathtt{lines}}(x,y)$ and $\widehat{A}(z)$, where z is a point on the line connecting x and y, is checked by considering the corresponding positions in $\Pi'_{\mathtt{lines}}((x,y),\cdot)$ and $\widehat{A}'(z,\cdot)$.

Turning to Step 2, recall that here we should obtain the product of three values of $\widehat{A} : \mathcal{F}^m \to \mathcal{F}$, and compare the result to a value obtained from Π_{4m} (i.e., Eq. (4) should equal the value of $\Pi_{4m}(r_1, ..., r_{4m-1})$ at r_{4m}). Recall that we wish to make a constant number of binary queries (and so we cannot afford querying $\widehat{A}$). On the other hand, it is unclear how we can get a linear combination of the bits of $\widehat{A}(x) \cdot \widehat{A}(z) \cdot \widehat{A}(z)$ by querying $\widehat{A}'$ at a constant number of places.

Specifically, for $a \in \mathcal{F}$ and $j \in [t]$, letting a_j denote the j^{th} bit of a, we observe that the i^{th} bit of $\widehat{A}(x) \cdot \widehat{A}(z) \cdot \widehat{A}(z)$ equals $\sum_{(j_1,j_2,j_3)\in S_i} \widehat{A}(x)_{j_1} \cdot \widehat{A}(z)_{j_2} \cdot \widehat{A}(z)_{j_3}$ for some $S_i \subseteq [t] \times [t] \times [t]$. However, we "don't have control" on these S_i's, and so it is unclear how linear combinations of the bits of

$\widehat{A}(x) \cdot \widehat{A}(z) \cdot \widehat{A}(z)$ can be recovered, in general, from a constant number of linear combination of the bits of $\widehat{A}(x)$, $\widehat{A}(y)$ and $\widehat{A}(z)$.

To address the foregoing problem, we just introduce an oracle that solves it. Specifically, consider the proof-oracle $T : (\mathcal{F}^m)^3 \times 2^{[t]^3} \to \{0,1\}$ such that for every $S \subseteq [t]^3$ it holds that

$$T(x,y,z,S) = \sum_{(i,j,k)\in S} \widehat{A}(x)_i \cdot \widehat{A}(y)_j \cdot \widehat{A}(z)_k \tag{5}$$

and note that $T(x,y,z,\cdot)$ is the Hadamard encoding of the t^3-bit long sequence

$$(\widehat{A}(x)_i \cdot \widehat{A}(y)_j \cdot \widehat{A}(z)_k)_{(i,j,k)\in[t]^3}$$

(which we view as a 3-dimensional tensor (of side-length t)). Hence, testing that a given W, which is claimed to equal $T(x,y,z,\cdot)$, actually equals it (i.e., satisfies Eq. (5) for every S) reduces to testing that W is a valid Hadamard codework and that the sequence that W encodes equals $(\widehat{A}(x)_i \cdot \widehat{A}(y)_j \cdot \widehat{A}(z)_k)_{(i,j,k)\in[t]^3}$. The second test is akin to the matrix equality test (of [2]). Details follow.

The second test is performed by viewing the message $w \in [t]^3$ encoded in W as a 3-dimensional tensor (of side-length t) and testing it against the tensor $(\widehat{A}(x)_i \cdot \widehat{A}(y)_j \cdot \widehat{A}(z)_k)_{(i,j,k)\in[t]^t}$. The test is performed by taking a corresponding random sub-cube of each of the two tensors; that is, for uniformly selected $R_1, R_2, R_3 \subseteq [t]$, we compare $W(R_1 \times R_2 \times R_3)$, which we obtain via self-correction on W, to

$$\sum_{(i,j,k)\in R_1\times R_2\times R_3} \widehat{A}(x)_i \cdot \widehat{A}(y)_j \cdot \widehat{A}(z)_k = \left(\sum_{i\in R_1} \widehat{A}(x)_i\right) \cdot \left(\sum_{j\in R_2} \widehat{A}(y)_j\right) \cdot \left(\sum_{k\in R_3} \widehat{A}(z)_k\right) \tag{6}$$

where each factor (on the r.h.s of Eq. (6)) is obtained from the corresponding value of $\widehat{A}'$ (i.e., $\sum_{i\in R} \widehat{A}(x)_i$ equals the inner-product of $\widehat{A}(x)$ and $(\rho_1, ..., \rho_t)$, where $\rho_j = 1$ if $j \in R$ and $\rho_j = 0$ otherwise).

3 Low-level description

The following detailed description refer to the high-level description provided in Section 2; hence, we strongly recommend reading Section 2 before reading the current section.

Preliminaries. For an ℓ-long sequence α over Σ, where in most cases $\Sigma = \mathcal{F}$ (but in one case we shall use $\Sigma = \{0,1\}$), and $i \in [\ell]$, we let α_i denote the i^{th} element in $\alpha = (\alpha_1, ..., \alpha_\ell)$. The empty sequence is denoted λ. For equal-length bit strings x and s, we denote by $\langle x, s\rangle$ their inner-product mod 2.

For $\alpha \in \mathcal{F} = \mathrm{GF}(2^t)$, we denote by M_α the t-by-t Boolean matrix that represents multiplication by α; that is, for every $x \in \mathcal{F}$, viewed as a t-bit long column vector, it holds that $M_\alpha x$ represents the element $\alpha \cdot x \in \mathcal{F}$. Note that $M_\alpha + M_\beta = M_{\alpha+\beta}$ and $M_\alpha M_\beta = M_{\alpha\beta}$. Also, viewing x as a t-bit long column vector and s as a t-bit long row vector, we have $\langle M_\alpha x, s\rangle = sM_\alpha x = \langle x, sM_\alpha\rangle$.

As stated in Footnote 4, we shall assume that the Boolean formula ϕ is monotone. This assumption can be justified by introducing a variable for each original literal, and checking consistency

of these pairs of variables (in addition in testing the satisfiability of the resulting monotone formula). We may also assume, without loss of generality, that the 3CNF formula has n variables and n clauses. In addition, we represent `true` (resp., `false`) by 0 (resp., by 1); this allows us to use $\widehat{A}(x) \cdot \widehat{A}(y) \cdot \widehat{A}(z)$ rather than $(1-\widehat{A}(x)) \cdot (1-\widehat{A}(y)) \cdot (1-\widehat{A}(z))$ in Eq. (1) (and in Eq. (7)). (Alternatively, we can retain the standard representation for A and define $\widehat{A}$ to be a lw-degree extension of $1-A$.)

Another issue that was avoided for simplicity (see also Footnote 4) is that Eq. (1) is inaccurate. The actual expression includes a pseudorandom sequence of coefficients $(\omega_\xi)_{\xi \in H^m}$ that are generated based on a random seed $\omega \in \{0,1\}^{O(\log n)}$, which is selected by the verifier. Wishing to simplify the exposition, we replace $\omega_\xi \cdot \Phi(\xi,x,y,z) \cdot \widehat{A}(x) \cdot \widehat{A}(y) \cdot \widehat{A}(z)$ by $\Phi_\omega(\xi,x,y,z) \cdot \widehat{A}(x) \cdot \widehat{A}(y) \cdot \widehat{A}(z)$. Hence, Eq. (1) is replaced by

$$\sum_{\xi,x,y,z \in H^m} \Phi_\omega(\xi,x,y,z) \cdot \widehat{A}(x) \cdot \widehat{A}(y) \cdot \widehat{A}(z), \tag{7}$$

where $\Phi_\omega : \mathcal{F}^{4m} \to \mathcal{F}$ is an explicit polynomial of individual degree $h-1$ that depends on ϕ (and ω, where $\omega \in \{0,1\}^{O(\log n)}$ is selected uniformly at random by the verifier). Furthermore, for sake of simplicity, we omit ω from the notation; that is, we use Φ rather than Φ_ω. More importantly, we use the notation Π_j although Π_j depends on ω. In other words, actually, we should have a different oracle $\Pi_{\omega,j}$ for each $\omega \in \{0,1\}^{O(\log n)}$, but for simplicity we continue using the notation Π_j.

The proof-oracle that we use. For a constant $m \in \mathbb{N}$, we view the input-oracle $A : [n] \to \{0,1\}$ as defined over $H^m \equiv [n]$, where $|H| = n^{1/m}$. We shall use a field $\mathcal{F} = \mathrm{GF}(2^t) \supset H$ such that $|F| = \mathrm{poly}(|H|)$, and consider the low-degree extension of A, which is denoted $\widehat{A} : \mathcal{F}^m \to \mathcal{F}$. The proof-oracle will consist of four parts, each encoding A (equiv., $\widehat{A}$) in a different way; each of these encoding is a concatenation code in which the inner-code is the Hadamard code.

- A *Hadamard encoding of the symbols of* $\widehat{A}$; that is, we use $\widehat{A}' : \mathcal{F}^m \times \mathcal{F} \to \{0,1\}$ such that $\widehat{A}'(x,s) = \langle \widehat{A}(x), s \rangle$ for every $s \in \{0,1\}^t \equiv \mathcal{F}$, where $\widehat{A} : \mathcal{F}^m \to \mathcal{F}$ is a low-degree extension of A (i.e., $\widehat{A}$ has individual degree $h-1$, and $\widehat{A}(x) = A(x)$ for every $x \in H^m$).

- A *Hadamard encoding of the restriction of* $\widehat{A}$ *to lines*; that is, $\Pi'_{\mathtt{lines}} : (\mathcal{F}^m)^2 \times \mathcal{F}^{mh} \to \{0,1\}$ such that $\Pi'_{\mathtt{lines}}((x,y),s) = \langle \Pi_{\mathtt{lines}}(x,y), s \rangle$ for every $s \in \{0,1\}^{mht} \equiv \mathcal{F}^{mh}$, where $\Pi_{\mathtt{lines}}(x,y) \in \mathcal{F}^{mh}$ is a representation of the univariate polynomial that describes the values of $\widehat{A}$ on the line that connects x and y. Specifically, for $\alpha \in \mathcal{F}$, the value of $\Pi_{\mathtt{lines}}(x,y)$ at $z = x + \alpha y$ is given by $\sum_{i=0}^{m \cdot (h-1)} \Pi_{\mathtt{lines}}(x,y)_{i+1} \cdot \alpha^i$, and this value is supposed to equal $\widehat{A}(z)$. (Recall that $\Pi_{\mathtt{lines}}(x,y)_{i+1}$ denotes the coefficient of the i^{th} power (of the variable) in the univarite polynomial $\Pi_{\mathtt{lines}}(x,y)$, and that $z = x + \alpha y \in \mathcal{F}^m$ is the α^{th} point on the line connecting x and y.)

- *Hadamard encodings of answers provided in the sum-check procedure*: For each $j \in [4m]$, we use an encoding $\Pi'_j : \mathcal{F}^{j-1} \times \mathcal{F}^h \to \{0,1\}$ of $\Pi_j : \mathcal{F}^{j-1} \to \mathcal{F}^h$; that is, $\Pi'_j(\overline{r},s) = \langle \Pi_j(\overline{r}), s \rangle$ for every $s \in \{0,1\}^{ht} \equiv \mathcal{F}^h$, where $\Pi_j(r_1,...,r_{j-1}) \in \mathcal{F}^h$ is a representation of the univariate polynomial that equals the r.h.s of Eq. (2).[6]

[6] Recall that we actually use a different $\Pi'_{\omega,j}$, defined based on the corresponding $\Pi_{\omega,j}$, for each $\omega \in \{0,1\}^{O(\log n)}$. We omitted ω from these notations for the sake of simplicity.

- A *Hadamard encoding of three-way products of $\widehat{A}$*: Here we use an encoding $T : (\mathcal{F}^m)^3 \times 2^{[t]^3} \to \{0,1\}$ of bits in all three-way products of values of $\widehat{A}$; that is, for every $x, y, z \in \mathcal{F}^m$ and $s \in \{0,1\}^{t^3}$, it holds that $T(x,y,z,s) = \langle \tau(x,y,z), s \rangle$, where $\tau(x,y,z)_{i,j,k} = \widehat{A}(x)_i \cdot \widehat{A}(y)_j \cdot \widehat{A}(z)_k$ for every $i,j,k \in [t]$.

  Recall that $\mathcal{F} \equiv \{0,1\}^t$ and that each bit in $\widehat{A}(x) \cdot \widehat{A}(y) \cdot \widehat{A}(z) \in \mathcal{F}$ can be expressed as a linear combination of the bits in $\tau(x,y,z) \in \{0,1\}^{t^3}$.

The verification boils down to testing that each of the oracles satisfies its definition and that Eq. (4) holds, where Eq. (4) depends on the input formula ϕ. Details follow.

The verification process. For simplicity, in the following description, we abuse notation and denote by $\widehat{A}', \Pi'_{\texttt{lines}}, \Pi'_j$ and T the actual proof-oracles to which the verifier has access (rather than the prescribed encodings defined above). Recall that each of these proof-oracles is supposed to be a concatenated code in which the inner-code is the Hadamard code. We will subject each inner-code to a corresponding codeword test, and then use self-correction on it. Further, recall that the input to each of these oracles consists of a pair $(\overline{x}, s)$ such that $\overline{x}$ points to a specific Hadamard codeword and s is a location in that codeword. Hence, we define the following *testing and self-correction procedure*, denoted $\texttt{SC}^B$, where B is one of these oracles. On input $(\overline{x}, s)$, this procedure proceeds as follows.

- It test that $B(\overline{x}, \cdot)$ is a Hadamard codeword, and proceeds only if this test accepts.
- It selects uniformly $r \in \{0,1\}^{|s|}$ and returns the value $B(\overline{x}, s+r) - B(\overline{x}, r)$.

Denoting the answer of this procedure by $\texttt{SC}^B(\overline{x}; s)$, we stress that if the test rejects then the answer is a special failure symbol and the verifier that uses $\texttt{SC}^B$ halts rejecting. We now turn to the verifier itself, which proceeds as follows.

1. *The low-degree test*: The verifier selects uniformly at random $x, y \in \mathcal{F}^m$ and $\alpha \in \mathcal{F}$, and computes the t-by-t Boolean matrices M_{α^i} for every $i \in \{0,1,...,m \cdot (h-1)\}$. (Recall that M_α represents multiplication by α (i.e., $M_\alpha \zeta = \alpha\zeta$).) Then, the verifier selects $s \in \{0,1\}^t$ uniformly at random, viewing it as a t-bit long row vector, and checks that

$$\texttt{SC}^{\Pi'_{\texttt{lines}}}((x,y); (s, sM_\alpha, sM_{\alpha^2}, ..., sM_{\alpha^{m\cdot(h-1)}})) = \texttt{SC}^{\widehat{A}'}(x + \alpha y; s).$$

   (The l.h.s yields the self-corrected value of $\Pi'_{\texttt{lines}}((x,y), (s, sM_\alpha, sM_{\alpha^2}, ..., sM_{\alpha^{m\cdot(h-1)}}))$, which equals $\langle s, \sum_{i=0}^{m\cdot(h-1)} M_{\alpha^i} \Pi_{\texttt{lines}}(x,y)_{i+1} \rangle$, which in turn equals $\sum_{i=0}^{m\cdot(h-1)} \langle s, \alpha^i \cdot \Pi_{\texttt{lines}}(x,y)_{i+1} \rangle$, whereas the r.h.s. yields a self-corrected value of $\widehat{A}'(x+\alpha y, s) = \langle s, \widehat{A}(x+\alpha y) \rangle$.)[7]

2. *The iterative sum-check tests* (i.e., the tests performed in the iterations of the sum-check procedure): The verifier computes the t-by-t Boolean matrices M_{r^i} for every $r \in H$ and $i \in \{0,1,...,h-1\}$. Then, the verifier selects ($\omega \in \{0,1\}^{O(\log n)}$ and)[8] $s \in \{0,1\}^t$ uniformly at random, viewing it as a t-bit long row vector (and assuming that h is odd), and checks that

$$\texttt{SC}^{\Pi'_1}(\lambda; (s, sM_{\sum_{r\in H} r}, sM_{\sum_{r\in H} r^2}, ..., sM_{\sum_{r\in H} r^{h-1}})) = 0.$$

[7] Indeed, $\Pi_{\texttt{lines}}(x,y)_{i+1}$ is viewed as a column vector that represents an element of $\mathcal{F} = \mathrm{GF}(2^t)$. We comment that self-correction is employed to $\widehat{A}'$ only for the sake of testing that $\widehat{A}'(x+\alpha y, \cdot)$ is a Hadamard codeword.

[8] Recall that ω identifies the polynomial $\Phi_\omega : \mathcal{F}^{4m} \to \mathcal{F}$ that appears in Eq. (7), which is the expression that should evaluate to 0, and that we chose to replace Φ_ω (resp., $\Pi_{\omega,j}$) by Φ (resp., Π_j) for the sake of simplicity.

(Note that the l.h.s equals the inner-product mod 2 of s and the sum of the values of the polynomial $\Pi_1(\lambda)$ evaluated at all points in H. Indeed, this corresponds to the check performed in the first iteration of the check-sum process.)[9]

Next, the verifier selects $r_1, ..., r_{4m} \in \mathcal{F}$ uniformly at random, and checks that for every $j \in \{2, ..., 4m\}$ it holds that

$$\begin{aligned} &\mathtt{SC}^{\Pi'_j}((r_1, ..., r_{j-1}); (s, sM_{\sum_{r\in H} r}, sM_{\sum_{r\in H} r^2}, ..., sM_{\sum_{r\in H} r^{h-1}})) \\ = \;& \mathtt{SC}^{\Pi'_{j-1}}((r_1, ..., r_{j-2}); (s, sM_{r_{j-1}}, sM_{r_{j-1}^2}, ..., sM_{r_{j-1}^{h-1}})). \end{aligned}$$

(Indeed, this corresponds to the check performed in the j^{th} iteration of the check-sum process, and we may use the same s in all $4m$ iterations. Note that $\Pi_j(r_1, ..., r_{j-1})$ is evaluated at all points in H as well as at r_j (where for $j = 4m$ the latter evaluation is performed in Step 4).)

3. *Testing consistency of T with $\widehat{A}$*: For $r_1, ..., r_{4m}$ as determined by the sum-check procedure, let $x = (r_1, ..., r_m)$, $y = (r_{m+1}, ..., r_{2m})$, and $z = (r_{2m+1}, ..., r_{4m})$. The verifier select $u, v, w \in \{0,1\}^t$ uniformly at random, and checks that

$$\widehat{A'}(x, u) \cdot \widehat{A'}(y, v) \cdot \widehat{A'}(z, w) = \mathtt{SC}^T((x, y, z); u \otimes v \otimes w)$$

holds, where $u \otimes v \otimes w$ denotes the 3-dimensional tensor obtained by an outer-product of u, v and w. (Analogously to Eq. (6), note that $\langle \widehat{A}(x), u\rangle \cdot \langle \widehat{A}(y), v\rangle \cdot \langle \widehat{A}(z), w\rangle$ equals $\langle \widehat{A}(x) \otimes \widehat{A}(y) \otimes \widehat{A}(z), u \otimes v \otimes w\rangle$.)[10]

4. *The final sum-check test* (i.e., the test performed after the last iteration of the sum-check procedure): For each $i \in [t]$, define $S_i \subseteq [t]^3$ such that for every $\alpha, \beta, \gamma \in \mathcal{F}$ it holds that the i^{th} bit of $\alpha\beta\gamma$ equals $\sum_{(j_1,j_2,j_3)\in S_i} \alpha_{j_1}\beta_{j_2}\gamma_{j_3}$.

   The verifier computes $\alpha \leftarrow \Phi(r_1, ..., r_{4m})$, where Φ is constructed based on the input formula ϕ.

   The verifier selects $s \in \mathcal{F}$ uniformly at random and computes $S = \oplus_{i:s_i=1} S_i$, where $\oplus_{i\in I} S_i$ contains (j_1, j_2, j_3) if and only if (j_1, j_2, j_3) appears in an odd number of S_i's (with $i \in I$). Assuming that $\alpha \neq 0$, the verifier obtains $b \leftarrow \mathtt{SC}^T(r_1, ..., r_{4m}, S)$ and checks that

$$\mathtt{SC}^{\Pi'_{4m}}((r_1, ..., r_{4m-1}); (sM_{\alpha^{-1}}, sM_{\alpha^{-1}\cdot r_{4m}}, sM_{\alpha^{-1}\cdot r_{4m}^2}, ..., sM_{\alpha^{-1}\cdot r_{4m}^{h-1}})) = b.$$

[9] Specifically, we wish to verify that

$$\left\langle s, \sum_{r\in H}\sum_{i=0}^{h-1} M_{r^i}\Pi_1(\lambda)_{i+1} \right\rangle = 0$$

where the l.h.s equals

$$\left\langle s, \sum_{i=0}^{h-1}\sum_{r\in H} M_{r^i}\Pi_1(\lambda)_{i+1} \right\rangle = \left\langle s, \sum_{i=0}^{h-1} M_{\sum_{r\in H} r^i}\Pi_1(\lambda)_{i+1} \right\rangle.$$

The hypothesis that h is odd implies that $sM_{h\cdot r^0} = s$; for even h we would have had $sM_{h\cdot r^0} = 0$.

[10] There is no need to employ self-correction on A', since it is evaluated at a random location, whereas Step 1 includes a test that $\widehat{A'}$ consists of a sequence of Hadamard codewords.

(If $\alpha = 0$, then the foregoing is checked while replacing $M_{\alpha^{-1} \cdot r_{4m}^j}$ by $M_{r_{4m}^j}$ and replacing b by 0.)[11]

(Indeed, this test corresponds to the check-sum process, except that (in case of $\alpha \neq 0$) both sides of the constraint are multiplied by α^{-1}.)[12]

5. *Testing consistency with the input-oracle*: The verifier selects uniformly at random $x \in H^m$ and $s = (s', \sigma) \in \{0,1\}^t \equiv \{0,1\}^{t-1} \times \{0,1\}$, and checks that $\langle \widehat{A}(x), s \rangle = A(x)$ if $\sigma = 1$ and $\langle \widehat{A}(x), s \rangle = 0$ otherwise.

   (Indeed, we assume that $b \in \{0,1\} \subset \mathcal{F}$ is represented by the bit string $0^{t-1}b$. Hence, $\langle 0^{t-1}b, s'\sigma \rangle = \sigma \cdot b$.)

The last step is needed for a PCPP for the claim that A satisfies ϕ, but not for a PCP for the claim that ϕ is satisfiable. Actually, when given proximity parameter $\epsilon > 0$, Step 5 is executed $O(1/\epsilon)$ times. Each of the other steps is executed a constant number of times (so that passing these tests implies that the proof-oracles are sufficiently close to being what they are supposed to be). Hence, the query complexity of this PCPP is $O(m + \epsilon^{-1})$. Observing that the randomness complexity is dominated by the randomness used by the self-correction procedure for Π'_{4m}, which is $O(mht) = O(n^{1/m} \log n)$, we obtain.

Theorem: *The forgoing verification procedure constitutes a PCPP for 3SAT with query complexity $O(m + \epsilon^{-1})$ and randomness complexity $\widetilde{O}(n^{1/m})$.*

Guideline for proving the theorem: The key observation is that the foregoing verification procedure emulates the corresponding RM-based PCP system. Specifically, each of the steps checks a random linear combination of t Boolean constraints that replace an $\mathcal{F}$-linear constraint that is checked by the RM-based PCP system. Let us spell out the corresponding checks of the latter system.

1. *The low-degree test*: The verifier selects uniformly at random $x, y \in \mathcal{F}^m$ and $\alpha \in \mathcal{F}$, and computes the t-by-t Boolean matrices M_{α^i} for $i = 0, 1, ..., m \cdot (h-1)$. It then checks that

$$\sum_{i=0}^{m \cdot (h-1)} M_{\alpha^i} \Pi_{\mathtt{lines}}(x, y)_{i+1} = \widehat{A}(x + \alpha y)$$

   holds, where M_α represents multiplication by α (i.e., $M_\alpha \zeta = \alpha \zeta$).

[11] That is, in this case there is no need to query T. We just check that

$$\mathtt{SC}^{\Pi'_{4m}}((r_1, ..., r_{4m-1}); (s, sM_{r_{4m}}, sM_{r_{4m}^2}, ..., sM_{r_{4m}^{h-1}})) = 0$$

holds.

[12] That is, rather than comparing that $v_{r_1,...,r_{4m}}$ to $\alpha \cdot \widehat{A}(r_{m+1}, ..., r_{2m}) \widehat{A}(r_{2m+1}, ..., r_{3m}) \widehat{A}(r_{3m+1}, ..., r_{4m})$ (as in Eq. (4)) we compared $\alpha^{-1} v_{r_1,...,r_{4m}}$ to $\tau(r_1, ..., r_{4m})$ (which supposedly equals $\widehat{A}(r_{m+1}, ..., r_{2m}) \widehat{A}(r_{2m+1}, ..., r_{3m}) \widehat{A}(r_{3m+1}, ..., r_{4m})$, by Step 3).

2. *The iterative sum-check tests*: The verifier selects $r_1, ..., r_{4m} \in \mathcal{F}$ uniformly at random.[13] For sake of clarity, we distinguish between the first iterations and the later iterations. In the first iteration, the verifier checks that

$$\sum_{r \in H} \sum_{i=0}^{h-1} M_{r^i} \Pi_1(\lambda)_{i+1} = 0$$

holds. (Recall that $\Pi_1(\lambda) : \mathcal{F} \to \mathcal{F}$ is a univariate polynomial of degree $h-1$.)

For $j \in \{2, ..., 4m\}$, in the j^{th} iteration, the verifier checks that

$$\sum_{r \in H} \sum_{i=0}^{h-1} M_{r^i} \Pi_j(r_1, ..., r_{j-1})_{i+1} = \sum_{i=0}^{h-1} M_{r_{j-1}^i} \Pi_{j-1}(r_1, ..., r_{j-2})_{i+1}$$

holds. (Recall that $\Pi_j(r_1, ..., r_{j-1}) : \mathcal{F} \to \mathcal{F}$ is a univariate polynomial of degree $h-1$.)

3. *Testing consistency of T with $\widehat{A}$*: Omitted. See next step.

4. *The final sum-check test*: The verifier computes $\alpha \leftarrow \Phi(r_1, ..., r_{4m})$, and checks that

$$\sum_{i=0}^{h-1} M_{r_{4m}^i} \Pi_{4m}(r_1, ..., r_{4m-1})_{i+1} = \alpha \cdot \widehat{A}(r_{m+1}, ..., r_{2m}) \cdot \widehat{A}(r_{2m+1}, ..., r_{3m}) \cdot \widehat{A}(r_{3m+1}, ..., r_{4m})$$

holds.

5. *Testing consistency with the input-oracle*: The verifier selects uniformly at random $x \in H^m$ and checks that $\widehat{A}(x) = A(x)$.

Regarding the iterative sum-check step, note that (analogously to Footnote 9), for every $j \in [4m]$, it holds that

$$\begin{aligned} \sum_{r \in H} \sum_{i=0}^{h-1} M_{r^i} \Pi_j(r_1, ..., r_{j-1})_{i+1} &= \sum_{i=0}^{h-1} \sum_{r \in H} M_{r^i} \Pi_j(r_1, ..., r_{j-1})_{i+1} \\ &= \sum_{i=0}^{h-1} M_{\sum_{r \in H} r^i} \Pi_j(r_1, ..., r_{j-1})_{i+1} \end{aligned}$$

4 Discussion

The question addressed here is whether or not it is fair to say that our construction does not use the proof composition paradigm. Towards addressing this question, we note that our construction differs from the RM-based PCP in two fundamental aspects:

[13] Recall that the verifier also selects $\omega \in \{0,1\}^{O(\log n)}$ uniformly at random and that Π_j actually stands for $\Pi_{\omega,j}$.

1. The proof-oracles used by our verifier are not RM-codewords but are rather codewords of concatenated codes. Furthermore, the proof-oracle T is a codeword of a rather "weird" code (i.e., it is obtained by concatenating a weird outer code with the Hadamard code).

2. On top of performing the sum-check procedure and low-degree tests (as the RM-based PCP does), our verifier also performs a consistency test that is akin to the one performed by the H-based PCP.

In our opinion, using a "weird" (or rather unnatural) code and using a procedure that was used in a different PCP system does not mean that one employs the proof composition technique. For sure, we do not invoke any general proof composition result (e.g., [3, 2, 5, 7]), but rather refer to the specific test that we wish to perform (i.e., correctness of multiplication over $\mathcal{F}$). Furthermore, we do not employ a preparatory parallelization step.[14]

However, we view the foregoing debate as quite scholastic. Terms such as "using X", "characterization", "combinatorial", "simple", "natural" and "intuitive" are all vague and undefined. Still, it does not mean that we should not care about the underlying notions; it just means that there is no point in arguing about them. Even saying that "our construction is different from known ones" falls within this category, and still we said so.

Acknowledgements

I am grateful to Madhu Sudan for useful and interesting discussions.

References

[1] P. Amireddy, A.R. Behera, S. Srinivasan, M. Sudan, and S.V. Willumsgaard. Ideals, Gröbner Bases, and PCPs. *ECCC*, TR25-165, 2025.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *Journal of the ACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[3] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *Journal of the ACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[4] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *23rd ACM Symposium on the Theory of Computing*, pages 21–31, 1991.

[5] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of Proximity, Shorter PCPs, and Applications to Coding. *SIAM Journal on Computing*, Vol. 36 (4), pages 889–974, 2006. Extended abstract in *36th STOC*, 2004.

[6] I. Dinur. The PCP Theorem by Gap Amplification. In *38th ACM Symposium on the Theory of Computing*, pages 241–250, 2006.

[14] The latter assertion can be made also with respect to Dinur's gap amplification technique [6]. However, unlike us, Dinur treats the computation that she wishes to verify as generic (since it is a large set of constraints over a large alphabet).

[7] I. Dinur and O. Reingold. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. *SIAM Journal on Computing*, Vol. 36 (4), pages 975–1024, 2006. Extended abstract in *45th FOCS*, 2004.

[8] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *Journal of the ACM*, Vol. 43, pages 268–292, 1996. Preliminary version in *32nd FOCS*, 1991.

[9] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[10] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. *Journal of the ACM*, Vol. 39, No. 4, pages 859–868, 1992. Preliminary version in *31st FOCS*, 1990.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il