2012 IEEE 27th Conference on Computational Complexity

Combinatorial PCPs with Short Proofs

Or Meir
Department of Computer Science
Stanford University
Stanford, CA, USA
ormeir@cs.stanford.edu

***Abstract*—The PCP theorem (Arora et. al., J. ACM 45(1,3)) asserts the existence of proofs that can be verified by reading a very small part of the proof. Since the discovery of the theorem, there has been a considerable work on improving the theorem in terms of the length of the proofs, culminating in the construction of PCPs of quasi-linear length, by Ben-Sasson and Sudan (SICOMP 38(2)) and Dinur (J. ACM 54(3)).**

One common theme in the aforementioned PCP constructions is that they all rely heavily on sophisticated algebraic machinery. The aforementioned work of Dinur (J. ACM 54(3)) suggested an alternative approach for constructing PCPs, which gives a simpler and arguably more intuitive proof of the PCP theorem using combinatorial techniques. However, this combinatorial construction only yields PCPs of polynomial length, and is therefore inferior to the algebraic constructions in this respect. This gives rise to the natural question of whether the proof length of the algebraic constructions can be matched using the combinatorial approach.

In this work, we provide a combinatorial construction of PCPs of length $n \cdot (\log n)^{O(\log \log n)}$, coming very close to the state of the art algebraic constructions (whose proof length is $n \cdot (\log n)^{O(1)}$). To this end, we develop a few generic PCP techniques which may be interesting in their own right.

It should be mentioned that our construction does use low degree polynomials at one point. However, our use of polynomials is confined to the construction of error correcting codes with a certain simple multiplication property, and it is conceivable that such codes can be constructed without the use of polynomials.

***Keywords*-Computational and artificial intelligence; Computational intelligence; Computation theory; Computational complexity**

I. Introduction

A. Background and Our Results

The PCP theorem [1], [2] is one of the major achievements of complexity theory. A PCP (Probabilistically Checkable Proof) is a proof system that allows checking the validity of a claim by reading only a constant number of bits of the proof. The PCP theorem asserts the existence of PCPs of polynomial length for any claim that can be stated as membership in an $\mathbf{NP}$ language. The theorem has found many applications, most notably in establishing lower bounds for approximation algorithms.

The original proof of the PCP theorem by Arora et al. [1], [2] was based on algebraic techniques: Given a claim to be verified, they construct a PCP for the claim by first "arithmetizing" the claim, i.e., reducing the claim to a related "algebraic" claim about polynomials over finite fields, and then constructing a PCP for this algebraic claim. The PCP for the algebraic claim, in turn, requires an arsenal of tools that employ the algebraic structure of polynomials. While those algebraic techniques are very important and useful, it seems somewhat odd that one has to go through algebra in order to prove the PCP theorem, since the theorem itself does not refer to algebra. Furthermore, those techniques seem to give little intuition as to why the theorem holds.

Given this state of affairs, it is an important goal to gain a better understanding of the PCP theorem and the reasons for which it holds. In her seminal paper, Dinur [3][1] has made a big step toward achieving this goal by giving an alternative proof of the PCP theorem using a combinatorial approach. Her proof is not only considerably simpler than the original proof, but also seems to shed more light on the intuitions that underlie the theorem.

However, Dinur's PCP construction is still inferior to the algebraic constructions in a few aspects. We believe that it is important to try to come up with combinatorial constructions of PCPs that match the algebraic constructions in those aspects, as this will hopefully advance our understanding of the PCP theorem. Two of those aspects, namely the running time of the verification procedure and the soundness error, have been dealt with in previous works on the subject [6], [7]. In this work, we deal with a third aspect which concerns the length of the proofs, to be discussed next.

Let L be a language in $\mathbf{NP}$, and recall that there is a polynomial-time algorithm V that verifies the membership of a string x in L when given an additional $\mathbf{NP}$-witness. Let $t : \mathbb{N} \to \mathbb{N}$ denote the running time of V. The original PCP theorem asserts that in order to verify that $x \in L$, the PCP verifier needs to use a proof of length $\mathrm{poly}\,(t(|x|))$. However, using algebraic techniques, one can construct PCP verifiers that use a proof of length only $t \cdot \mathrm{poly}\log(t)$ [8], [3]. It is not known whether one can construct such PCPs using a combinatorial approach such as Dinur's[2].

[1]We mention that the works of [4], [5] have addressed this goal prior to Dinur's work [Din07], but fell short of obtaining Dinur's result.

[2]We mention that the construction of PCPs that have proof length $t \cdot \mathrm{poly}\log(t)$ uses Dinur's combinatorial techniques in addition to the algebraic techniques. Still, the main part of this construction is algebraic.

1093-0159/12 $26.00 © 2012 IEEE
DOI 10.1109/CCC.2012.14

In this work, we present an (almost) combinatorial construction of PCPs that use proofs of length $t \cdot (\log t)^{O(\log \log t)}$, thus coming very close to the state of the art algebraic constructions. Namely, our main result is the following

Theorem I.1 (Main theorem)**.** *For every time-constructible $t : \mathbb{N} \to \mathbb{N}$ and every language $L \in \mathbf{NTIME}(t)$, there exists a PCP verifier for L with proof length $t(n) \cdot (\log (t(n)))^{O(\log \log t(n))}$, query complexity $O(1)$, and rejection probability $\Omega(1)$.*

In order to prove Theorem I.1, we develop a few generic PCP techniques that may be of independent interest, and are discussed in Section I-B.

Our use of polynomials: As mentioned above, our construction is only "almost" combinatorial. The only exception is that our construction does use low degree polynomials at one point. However, our use of polynomials is a very restricted one, and is confined to the construction of error correcting codes with a certain simple property. Specifically, we only use polynomials to construct a triplet (C_A, C_B, C_M) of linear codes that have the following "multiplication property":

- For every two codewords $c_A \in C_A$ and $c_B \in C_B$, it holds that $c_A \cdot c_B$ is a codeword of C_M, where $c_A \cdot c_B$ is obtained by coordinate-wise multiplication of c_A and c_B. The multiplication is done in the finite field over which the codes are linear.

Such a triplet (C_A, C_B, C_M) can easily be constructed using low-degree polynomials. Moreover, if the codes C_A, C_B, and C_M are allowed to have quadratic length, then they can also be constructed without using polynomials [9]. However, for our purposes we need the codes C_A, C_B, and C_M to have quasi-linear length, and we do not know how to construct such codes without using polynomials. Still, a combinatorial construction of such codes is conceivable.

Extensions of our result: It can be shown that our PCPs have randomness complexity of $\log t + O(\log^2 \log t)$, which matches the length of the proofs. In addition, as in previous works in the area, our construction of PCPs can be extended to yield the stronger notion of PCPPs [10], [5]. We do not elaborate on those claims in this extended abstract.

B. Our techniques

Below, we sketch the main steps of our construction and the main techniques that we use.

Constructing PCPs from linear PCPPs: Our first step is reducing the construction of PCPs to the construction of a simpler object, called linear PCPPs [11]. Informally, a linear PCPP is a verifier that, when given a linear subspace $W \subseteq \mathbb{F}^n$ and oracle access to a vector $w \in \mathbb{F}^n$, verifies that $w \in W$ by making few queries to w and to an alleged proof. In other words, a linear PCPP is the restriction of the notion of a PCPP [10], [5] to the verification of membership in linear subspaces.

We show that *any* construction of a linear PCPP implies a construction of a full-fledged PCP, with a poly-logarithmic loss in the parameters. The construction of the full-fledged PCP is generic, and uses the linear PCPP as a black box. We believe that this construction is interesting in its own right, and may be useful for future works[3].

Our construction of PCPs from linear PCPPs is performed by combining the linear PCPP with the multiplication codes that were discussed in Section I-A. Intuitively, the multiplication codes allow us to go from verifying linear claims to verifying non-linear claims.

Robustization via tensor product codes: Our next step is to note, following [10], [5], [8], [3], that it suffices to construct a linear PCPP that makes $\tilde{O}(\sqrt{n})$ queries to its oracle and has proof length $\tilde{O}(n)$. Such a linear PCPP can then be composed with itself for $O(\log \log n)$ times to yield[4] a linear PCPP with a constant number of queries and proof length $n \cdot (\log n)^{O(\log \log n)}$.

In order for us to be able to apply such composition, the linear PCPP is required to have a property called robustness [10], [5]. The robustness property was achieved in several previous works by a technique called "robustization" [10], [5] (a.k.a. "alphabet reduction" or "parallelization"). The robustization technique allows one to transform every PCP with a "block-access" property into a robust PCP. Specifically, the latter property may be viewed as follows: It is required that the PCP proof can be partitioned to blocks, such that the PCP verifier always queries only a constant number of the blocks.

In our context, we do not know how to construct a PCP that satisfies the foregoing block-access property. We therefore generalize the robustization technique such that it can be applied to PCPs that satisfy a weaker requirement.

To this end, instead of partitioning the proof to blocks, we arrange the proof coordinates in a matrix. From this point of view, the foregoing block-access property may be viewed as restricting the PCP verifier to reading a constant number of rows of the matrix. We now generalize the robustization technique by allowing the PCP verifier to query *both rows and columns* of the aforementioned matrix, as long as it queries a *constant number* of rows and columns.

[3] We note that the work of [8] has shown a stronger result, namely that one can construct a full-feldged PCP from a linear PCPP that *can only verify membership in a Reed-Solomon code* (rather than a general linear subspace). However, their construction is signicantly more complicated than ours, and relies heavily on algebraic machinery.

We also note that other works in this area (e.g. [1], [2], [10]) start by reducing the construction of a full-edged PCP to the construction of a PCP for a specific algebraic problem. However, in all of those works, the latter algebraic problem is $\mathbf{NP}$-complete, while the problem we consider (checking membership in a linear subspace) is in $\mathbf{P}$. Hence, there is a fundamental dierence between those reductions and ours.

[4] We mention that after the composition one also needs to apply a query reduction technique and the gap amplification theorem of Dinur.

Both the standard robustization technique and our generalization use error correcting codes. The standard robustization technique transforms a PCP that has "block-access" into a robust one by encoding each of the blocks by an error correcting code. In our a generalization, we transform the "row/column-access PCP" into a robust one by encoding the corresponding matrix via a robust tensor product code [12]. This means, roughly, that we first encode the rows of the matrix by an error correcting code, and then encode the columns of the new matrix by an error correcting code.

We stress that this generalization of the robustization method is generic, and may be useful for future constructions of PCPs.

Constructing linear PCPPs that make $\tilde{O}(\sqrt{n})$ queries: It remains to construct a linear PCPP that makes $\tilde{O}(\sqrt{n})$ queries to its oracle, has proof length $\tilde{O}(n)$, and satisfies the relaxed robustization requirement discussed above. The linear PCPP that we construct works verifies a linear assetion in two stages. In the first stage, the tested linear assertion is decomposed into $\tilde{O}(\sqrt{n})$ smaller linear assertions of size $\tilde{O}(\sqrt{n})$, such that the original assertion holds if and only if all the smaller assertions hold simultaneously. This is done using a decomposition technique of [7].

In the second stage, the linear PCPP verifies that all the smaller assertions hold simultaneously. To this end, we begin by considering the special case in which all the smaller linear assertions are identical, and show how to handle this case using error correcting codes. Then, we show how to decompose the general case to a constant number of instances of the foregoing special case, and handle those instances as before. The latter decomposition is performed via a novel application of routing networks and of the multiplication codes discussed in Section I-A.

In the author's opinion, the second stage of this construction is the most interesting part of this work.

Organization of this paper: In Section II, we review the preliminaries that are required for this work. In Section III, we define the notion of linear PCPPs, and show how to construct a PCP based on a linear PCP. In Section IV, we review the standard robustization technique, and sketch our generalization of this technique. In Section V, we present the construction of linear PCPPs with $\tilde{O}(\sqrt{n})$ queries. Finally, in Section VI, we show how to use the foregoing tools to construct the required PCPs and prove the main theorem (Theorem I.1).

II. Preliminaries

For any $n \in \mathbb{N}$ we denote $[n] \stackrel{\text{def}}{=} \{1 \ldots, n\}$. For a string $x \in \{0,1\}^n$ and a set $S \subseteq [n]$, we denote by $x_{|S}$ the projection of x to the coordinates in S.

A. PCPs

Below we give the formal definitions of PCPs.

Definition II.1. Let $L \subseteq \{0,1\}^*$ be a language, and let $q, \ell : \mathbb{N} \to \mathbb{N}$, $\rho : \mathbb{N} \to (0,1)$. A PCP verifier V for L with query complexity q, proof length ℓ, and rejection probability ρ, is a probabilistic oracle machine that satisfies the following requirements:

1) On every input $x \in \{0,1\}^*$ and every sequence of coin tosses, V makes at most $q(|x|)$ queries to its oracle.
2) **Completeness:** For every $x \in L$, there exists a string $\pi \in \{0,1\}^{\ell(|x|)}$ such that $\Pr[V^\pi(x) \text{ accepts}] = 1$.
3) **Soundness:** For every $x \notin L$ and every a string $\pi \in \{0,1\}^{\ell(|x|)}$, it holds that $\Pr[V^\pi(x) \text{ rejects}] \geq \rho(|x|)$.

B. Error Correcting Codes

All the error correcting codes that we consider in this extended abstract are *binary linear codes*, to be defined next. A (linear) code C with message length k and block length ℓ is an injective linear function from $\{0,1\}^k$ to $\{0,1\}^\ell$ (where linearity is over GF(2)). We will sometimes identify C with its image $C(\{0,1\}^k)$. Specifically, we will write $c \in C$ to indicate the fact that there exists $x \in \{0,1\}^k$ such that $c = C(x)$. In such case, we also say that c is a codeword of C. The rate R_C of the code C is the ratio k/ℓ.

For any two strings $x, y \in \{0,1\}^\ell$, the relative Hamming distance between x and y is the fraction of coordinates on which x and y differ, and is denoted by $\delta(x,y) \stackrel{\text{def}}{=} |\{x_i \neq y_i : i \in [\ell]\}| / \ell$. The relative distance of a code C is defined as $\delta_C \stackrel{\text{def}}{=} \min_{c_1 \neq c_2 \in C} \{\delta(c_1, c_2)\}$. For every code $C : \{0,1\}^k \to \{0,1\}^\ell$ and a string $w \in \{0,1\}^\ell$ we denote $\delta(w, C) \stackrel{\text{def}}{=} \min_{c \in C} \{\delta(w, c)\}$. We say that w is ε-far from C (resp. ε-close to C) if $\delta(w, C) > \varepsilon$ (resp. $\delta(w, C) \leq \varepsilon$).

A code $C : \{0,1\}^k \to \{0,1\}^\ell$ is said to be systematic if for every $x \in \{0,1\}^k$ it holds that $C(x)_{|[k]} = x$. By Gaussian elimination, every linear code may be assumed to be systematic without loss of generality. All the codes in this paper are assumed to be systematic.

C. Routing networks

In our construction of PCPs, we use a special kind of graphs called permutation routing networks (see, e.g., [13]). In order to motivate this notion, let us think of the vertices of the graph as computers in a network, such that two computers can communicate if and only if they are connected by an edge. Suppose that there is some set S of computers in the network such that each computer in S needs to send a message to some other computer in S, and furthermore that each computer in S needs to receive a message from exactly one computer in S (in other words, the mapping from source computers to target computers is a permutation). Then, the property of the routing network says that we can route the messages in the network such that each computer in the network forwards exactly one message. Formally, we use the following definition of routing networks.

Definition II.2. A routing network of order n is a graph $G = (V, E)$ along with a special set of vertices $S \subseteq V$ of size n, such that the following requirement holds: For every permuation σ on S, there exists a set $\mathcal{P}$ of vertex-disjoint paths in G that connect each $v \in S$ to $\sigma(v) \in s$.

Routing networks were studied extensively in the literature of distributed computing, and several constructions of efficient routing networks are known. In particular, we use the following fact on routing networks, whose requirements are satisfied by several constructions.

Fact II.3 (see, e.g, [13])**.** *There exists an infinite family of routing networks $\{G_n\}_{n=1}^{\infty}$, the network G_n being of order n such that the following properties hold.*

1) *G_n has $\tilde{O}(n)$ vertices.*
2) *The edges of G_n can be colored using 4 colors such that no two edges of the same color share a vertex.*
3) *There exists an algorithm that on input n, runs in time $\mathrm{poly}(n)$ and outputs G_n.*
4) *There exists a polynomial time algorithm that when given as input G_n and a permutation $\sigma : S \to S$ outputs a set $\mathcal{P}$ of vertex-disjoint paths that connect each $v \in S$ to $\sigma(v)$.*

III. PCPs and Linear PCPPs.

In this section we define the notion of linear PCPPs, and show how to construct a full-fledged PCP using a linear PCPP as a building block.

A. Linear PCPPs

We begin by defining the notion of linear PCPPs (originally defined in [11]). For simplicity, in this extended abstract we restrict our attention to linear PCPPs over $\mathrm{GF}(2)$, but it is possible to generalize this definition to larger fields, and in fact we do use such a generalization in the actual proof.

Informally, a linear PCPP is a verifier that checks that a vector w satisfies a linear assertion by reading a small part of w, and of an alledged proof. Little more specifically, a linear PCPP is an oracle machine that takes as explicit input a linear subspace $W \subseteq \{0,1\}^m$, gets oracle access to a vector $w \in \{0,1\}^m$ and to an additional string $\pi \in \{0,1\}^*$, accepts with probability 1 if w belongs to W, and rejects with significant probability if w is far from W. The subspace W is represented by a *linear circuit* [14], to be defined next.

A linear circuit (over $\mathrm{GF}(2)$) is a circuit that contains only gates that compute the XOR of their inputs[5]. The size of the circuit is defined to be the number of wires in the circuit. Note that every output of the circuit is a linear *non-affine* function[6] of the inputs, and that every linear function over $\mathrm{GF}(2)$ can be computed by such circuits. We note that in some previous works the definition of linear circuits is little different, and allows the circuits to compute affine functions. The reason we choose not to allow affine functions is that it allows us to state a stronger result (Theorem III.6).

We will usually consider linear circuits that have multiple outputs. We say that a linear circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ accepts an input $w \in \{0,1\}^m$ if $\varphi(w) = \overline{0} \in \{0,1\}^t$. Observe that the set of inputs accepted by a linear circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ is a linear subspace of $\{0,1\}^m$ of dimension at least $m - t$. We denote the latter subspace by W_φ, and say that φ accepts W_φ.

Definition III.1 (Variant of [11])**.** Let $q, \ell : \mathbb{N} \to \mathbb{N}$, $\rho : \mathbb{N} \to (0,1)$. A linear PCPP verifier V with query complexity q, proof length ℓ, and rejection ratio ρ, is a probabilistic oracle machine that satisfies the following requirements:

1) V takes as input a linear circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ of size n, and gets oracle access to a vector $w \in \{0,1\}^m$ as well as to an additional string $\pi \in \{0,1\}^{\ell(n)}$.
2) On every input circuit φ and every sequence of coin tosses, V makes at total number of at most $q(n)$ queries to its oracle. The queries are made non-adaptively.
3) **Completeness:** For every $x \in W_\varphi$, there exists a string $\pi \in \{0,1\}^{\ell(n)}$ such that $\Pr[V^{x,\pi}(\varphi) \text{ accepts}] = 1$.
4) **Soundness:** For every $x \in \{0,1\}^m$ that is ε-far from W_φ and every string $\pi \in \{0,1\}^{\ell(n)}$, it holds that $\Pr[V^{x,\pi}(\varphi) \text{ rejects}] \geq \rho(n) \cdot \varepsilon$.

Remark III.2. In order to be able to compose linear PCPPs, we also need to require that after the linear PCPP gets the answers it gets to its queries, it may only apply *linear* predicates to those answers. However, we ignore this issue in this extended abstract.

Remark III.3. The notion of linear PCPP is a special case of the notion of PCPP of [10], [5] (a.k.a assignment tester). It is also related to the notion of linear inner verifier of [15].

B. Constructing PCPs from linear PCPPs

We turn to show a construction of PCPs that uses linear PCPPs as a building block. This is formalized in Theorem III.6 below.

Before getting to the theorem and its proof, we note that the proof relies on the existence of "multiplication codes", which are codes with a certain multiplication property. For simplicity, in this extended abstract we assume the existence of multiplication codes over $\{0,1\}$, while in the actual proof we use such codes over a larger field, and construct them using the Reed-Solomon code. More specifically, we assume the following:

[5] In the generalization of this definition to larger fields, every gate computes a linear combination of its inputs.

[6] The function is non-affine because we did not allow constant gates.

Assumption III.4. *For every $k \in \mathbb{N}$, there exists a triplet of systematic codes (C_A, C_B, C_M) of constant rate and relative distance, such that C_A and C_B have message length k, and such that the following holds: For every $c_A \in C_A$ and $c_B \in C_B$, it holds that $c_A \cdot c_B \in C_M$, where $c_A \cdot c_B$ is the coordinate-wise multiplication of c_A and c_B.*

Remark III.5. In fact, our construction would have worked even if the rate and distance of C_A, C_B, and C_M were only $1/\operatorname{poly}\log k$. However, we preferred to use constants for simplicity and because it can be achieved (over larger fields). While we only know how to construct such codes by using polynomials, it is plausible that such codes can be constructed in other ways.

In the rest of this section, we sketch the proof of the following theorem, based on Assumption III.4.

Theorem III.6. *Suppose that there exists a linear PCPP verifier V with query complexity $q(n)$, proof length $\ell(n)$, and rejection ratio $\rho(n)$. Then, for every time-constructible $t : \mathbb{N} \to \mathbb{N}$ and every language $L \in \mathbf{NTIME}(t)$, there exists a PCP verifier V' for L with proof length $\tilde{O}(t) + O\left(\ell\left(\tilde{O}(t)\right)\right)$ query complexity $O\left(q\left(\tilde{O}(t)\right) \cdot \log n\right)$, and rejection probability $\Omega\left(\rho\left(\tilde{O}(t)\right)\right)$.*

Remark III.7. The extra $\log n$ factor in the query complexity comes from the use of large fields in the construction of the multiplication codes, and is not apparent in this extended abstract. In our final construction of PCPs this $\log n$ factor is reduced in a later stage using known query reduction techniques.

Intuitively, the idea that underlies the proof of Theorem III.6 is that the multiplication codes allow us to go from verifying linear claims to verifying non-linear claims. Little more specifically, we begin by noting that it suffices to construct a PCP for the language of satisfiable systems of quadratic equations, since this is an $\mathbf{NP}$-complete language. Given a system of quadratic equations E, the PCP proof will be expected to contain a satisfying assignment x to the system, as well as a string y that should contain the value under x of each quadratic term that appears in E. Now, E may be viewed as a system of *linear* equations over x and y, so we can use the linear PCPP verifier V on x and y to check that it is satisfied. It remains to verify that y is indeed consistent with x, and the point is that this consistency can be verified by using the multiplication codes together with V.

To be more concrete, the verifier V' acts as follows. Let C_A, C_B, and C_M be as in Assumption III.4. The verifier V' constructs two projections of the string x, denoted a and b, such that $y = a \cdot b$ (where the multiplication is coordinate-wise). The verifier V' expects the proof to contain the encodings $a' \stackrel{\text{def}}{=} C_A(a)$ and $b' \stackrel{\text{def}}{=} C_B(b)$, as well as the vector $c' \stackrel{\text{def}}{=} C_A(a) \cdot C_B(b) \in C_M$. Note that the string y is a substring of c', since C_A, C_B, and C_M are systematic. Now, V' invokes V to verify that the system E is satisfied as a linear system over x and y, where y is retrieved from c'.

It remains for V to verify that the vectors a', b', and c' in the proof are constructed as expected. To this end, observe that $C_A(a)$ and $C_B(b)$ are obtained from x via a linear transformation, and hence V' verifies the consistency of a' and b' with x simply by invoking V. In order to verify the consistency of c', the verifier V' checks that the vectors c' and $a' \cdot b'$ agree on a random coordinate. The soundness of the latter check is proved using the relative distance of C_M.

Remark III.8. One may argue that using the language of quadratic equations is an "algebraic step". However, it is not hard to adjust the proof to work with the language CIRCUITSAT instead of quadratic equations. We chose the language of quadratic equations for technical convenience.

IV. Generalization of Robustization

Our construction of PCPs uses the composition technique in order to reduce the query complexity of our linear PCPPs (see details in Section VI). In order to apply composition, we need our linear PCPPs to have a property called robustness [10], [5]. As discussed in Section I-B, this property was achieved in previous works by a technique called "robustization", which can not be applied to our linear PCPPs. In order tor resolve this issue, we generalize the robustization technique such that it can be applied to our linear PCPPs.

In Section IV-A, we define the notion of robustness and describe the standard robustization method. In Section IV-B, we describe our generalization of the robustization method.

A. Background on robustness and robustization

Below we provide some background on the robustness property, and a more detailed information may be found in [10], [5]. All the definitions and results discussed here are stated in terms of linear PCPPs, although they were originally developed for PCPs. Informally, a linear PCPP verifier V is robust if, when V is given oracle access to a vector x that is far from satisfying the linear assertion that is being checked, the answers to V's queries are far from making V accept.

Definition IV.1 (Views and accepting views)**.** Let V be a linear PCPP verifier with proof length ℓ. Fix an input circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ of size n, a vector $x \in \{0,1\}^m$, and a proof string $\pi \in \{0,1\}^{\ell(n)}$. For every possible invocation of V, we refer to the answers that V gets to its queries as the view of V. If V accepts, then the corresponding view is said to be an accepted view.

Definition IV.2 (Robustness)**.** Let $\rho : \mathbb{N} \to (0,1)$, and let. V be a linear PCPP verifier. The verifier V is said to have robustness ρ if whenever x is ε-far from W_φ, the expected relative distance of the view of V from the closest accepted view is at least $\rho(n) \cdot \varepsilon$.

Remark IV.3. Observe that if a linear PCPP has robustness ρ then it in particular has rejection ratio ρ. Thus, when discussing robust linear PCPPs we do not mention their rejection ratio.

A common technique in the PCP literature, called "robustization" (also "alphabet reduction" or "parallelization"), allows transforming every PCP with a certain query structure into a robust PCP. Specifically, the technique requires the following property:

Definition IV.4. We say that a linear PCPP verifier V has κ-block access if for every circuit φ, the coordiantes of the oracle $x \circ \pi$ can be partitioned into blocks, such that V always queries at most κ blocks.

The robustization technique yields the following result.

Theorem IV.5 (Robustization, special case proved in [10], [5], [7])**.** *Suppose that there exists a linear PCPP verifier V with query complexity $q(n)$, proof length $\ell(n)$, and rejection ratio $\rho(n)$, which has κ-block access. Then, there exists a linear PCPP verifier V' with query complexity $O(q)$, proof length $O(\ell)$, and* robustness $\Omega(\rho/\kappa)$.

The verifier V' is constructed roughly as follows. The proof is expected to contain the encoding of each of the blocks via an error correcting code. The verifier V' emulates the verifier V, but whenever V queries a block, the verifier V' also queries the purported encoding of the block, and verifies that it is indeed the legal encoding of the block.

To see that V' is robust, observe that whenever V rejects, at least one of the blocks that V queries must be modified in order to make V accept. Hence, in order to make V' accept, the both the aforementioned block *and its encoding* must be modified. However, modifying the latter encoding to another legal encoding requires flipping many coordinates, and therefore the view of V' is far from any accepting view.

B. Our generalized robustization

Unfortunately, we do not know how to make our linear PCPPs to have κ-block access. In order to resolve this issue, we define a relaxation of the block access property, which we call row/column access, and prove a more general robustization theorem that applies to PCPs with the latter property.

Definition IV.6. We say that a linear PCPP verifier V has κ-row/column access if for every circuit φ and every strings x, π, the string $x \circ \pi$ can be arranged in a matrix M, such that the coordinates of $x \circ \pi$ that are queried by V consistitute whole rows and columns of M, and at most κ such rows and columns.

We now have the following result.

Theorem IV.7 (Generalized robustization)**.** *Suppose that there exists a linear PCPP verifier V with query complexity q, proof length ℓ, and rejection ratio ρ, which has κ-row/column access. Then, there exists a linear PCPP verifier V' with query complexity $O(q)$, proof length $O(\ell)$, and* robustness $\Omega(\rho/\kappa)$.

It is tempting to try to prove Theorem IV.7 using the same argument as for Theorem IV.5. Such a construction would require the proof to contain the encoding of every row and column of M via an error correcting code. Then, the verifier V' would read the encoding of every row and column that are queried by V and check that it is a legal encoding.

However, the foregoing argument fails. The reason is that the purported encodings of the rows and columns of M may be *inconsistent*. That is, the proof might contain encodings whose encoded messages do not agree on the intersections of the rows and columns. Such an inconsistency may fail the soundness of V'.

In order to resolve this issue, we use tensor codes, to be defined next.

Definition IV.8 (Tensor codes, see, e.g., [16, Lect. 6 (2.4)])**.** Let $C : \{0,1\}^k \to \{0,1\}^\ell$ be a code. The tensor code C^2 is a code with block length ℓ^2, whose codewords are exactly the $\ell \times \ell$ matrices N such that every row and every column of N is a codeword of C.

Fact IV.9. *Let $C : \{0,1\}^k \to \{0,1\}^\ell$ be a code of relative distance δ. Then, C^2 has message length k^2 and relative distance δ^2. Furtheremore, if C is systematic, then the message encoded by a codeword N of C^2 is the top-left $k \times k$ submatrix of N.*

The critical property of tensor codes that we use is the following: Let N' be a matrix that is close to a codeword N of C^2. Then, for most of the rows and columns of N', the closest codeword of C is the corresponding row or column of N. In particular, this means that the messages encoded by most rows and columns of N' are consistent with each other.

Now, we construct the verifier V' such that the proof is expected to contain the encoding of the matrix M via a tensor code C^2. The soundness analysis goes roughly as follows. Let N' be the purported encoding of M in the proof. We first note that if N' is close to C^2, then we can use the foregoing property of C^2 to argue that the rows and columns are mostly consistent, and perform roughly the same analysis as in Theorem IV.5.

It remains to verify that N' is close to C^2. To this end, we require C^2 to have a *robust tester*. This means, roughly, that there exists a *robust* linear PCPP verifier that is only capable

of verifying the assertion that a matrix is close to C^2 (rather than verifying any linear assertion). Tensor codes satisfying this requirement can be constructed combinatorially using a few methods (see [12], [17], [18], [19]). The verifier V' will invoke the robust tester of C^2 to verify that N' is close to C^2, and then proceed as before.

Remark IV.10. The foregoing description oversimplifies things a little. In particular, note that the property of C^2 only guarantees that *most of* the rows and columns are consistent, and not all of them. Thus, in general it could be the case that V always queries the inconsistent rows and columns. In order to handle this issue, we begin the construction by invoking the expander-replacement technique of [20], which guarantees that V queries rows and columns according to the uniform distribution, and then proceed as before.

V. Construction of Linear PCPPs with $\sqrt{n}$ Queries

In this section, we explain how to construct linear PCPPs that have proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and rejection ratio $1/\operatorname{poly}\log n$. In Section VI, we show that such linear PCPPs are sufficient for constructing the required PCPs.

In Section V-A below, we define an auxiliary object called simultaneous linear verifier (SLV). Then, in Section V-B, we describe how to construct the required linear PCPPs based on the existence of SLVs. Finally, in Section V-C, we show a construction of SLVs.

In the author's opinion, the construction of SLVs is the most interesting part in this work.

Remark regarding robustization: As discussed in the introduction, we will need our linear PCPPs to be robust. In this extended abstract we ignore this issue completely. However, the robustness can be achieved by modifying the construction by a little and applying the robustization technique of Section I-B.

A. Simultaneous linear verifiers

Intuitively, an SLV is a variant of a linear PCPP verifier, which verifies $\sqrt{n}$ linear assertions of size $\sqrt{n}$ instead of verifying one linear claim of size n. The verification is *simultaneous*, in the sense that if *at least one* of the $\sqrt{n}$ linear assertions is far from being correct, then the SLV rejects with noticeable probability.

To see the difference between SLVs and linear PCPPs, observe that if a linear PCPP verifier is invoked on a linear assertion of size n that consists of $\sqrt{n}$ concatenated linear assertions of size $\sqrt{n}$, then the verifier is required to reject only if the $\sqrt{n}$ linear assertions are far from being correct *on average* (i.e., a *random* assertion is far from being correct).

Definition V.1. Let $q, \ell : \mathbb{N} \to \mathbb{N}$, $\rho : \mathbb{N} \to (0,1)$. A simultaneous linear verifier (SLV) V with query complexity q, proof length ℓ, and rejection ratio ρ, is a probabilistic oracle machine that satisfies the following requirements:

1) V takes as input $\sqrt{n}$ linear circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}} : \{0,1\}^m \to \{0,1\}^t$ of size at most $\sqrt{n}$, and gets oracle access to $\sqrt{n}$ vectors $x_1, \ldots, x_{\sqrt{n}} \in \{0,1\}^m$ as well as to an additional string $\pi \in \{0,1\}^{\ell(n)}$.
2) For every input circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ and every sequence of coin tosses, V makes at most $q(n)$ non-adaptive queries to its oracle.
3) **Completeness:** For every $x_1 \in W_{\varphi_1}, \ldots, x_{\sqrt{n}} \in W_{\varphi_{\sqrt{n}}}$, there exists a string $\pi \in \{0,1\}^{\ell(n)}$ such that $\Pr\left[V^{x_1,\ldots,x_{\sqrt{n}},\pi} \text{ accepts}\right] = 1$.
4) **Soundness:** For every $x_1, \ldots, x_{\sqrt{n}} \in \{0,1\}^m$ such that for some $i \in [\sqrt{n}]$ it holds that x_i is ε-far from W_{φ_i}, and for every string $\pi \in \{0,1\}^{\ell(n)}$, it holds that $\Pr\left[V^{x_1,\ldots,x_{\sqrt{n}},\pi} \text{ rejects}\right] \geq \rho(n) \cdot \varepsilon$.

B. Linear PCPPs from simultaneous linear verifiers

As discussed above, the importance of SLVs is that we are able to construct a linear PCPP using an SLV as a building block. More specifically, we have the following result.

Theorem V.2. *Suppose that there exists an SLV V with query complexity $q(n)$, proof length $\ell(n)$, and rejection ratio $\rho(n)$. Then, there exists a linear PCPP verifier V' with query complexity $O\left(q\left(\tilde{O}(n)\right)\right)$, proof length $O\left(\tilde{O}(n) + \ell\left(\tilde{O}(n)\right)\right)$, and rejection probability $\Omega\left(\rho\left(\tilde{O}(n)\right)\right)$.*

The basic idea of the proof is to decompose the linear assertion that should be verified to a collection of $\tilde{O}(\sqrt{n})$ linear assertions of size $\tilde{O}(\sqrt{n})$ by using a decomposition method of [7], and then applying the SLV to verify the latter linear assertions. We note that the only novelty in the proof is the idea of using the technique of [7] in our setting (which is different than the setting of [7]). The technical part of the proof follows the technique of [7] quite closely.

C. Construction of simultaneous linear verifiers

In the rest of this section, we finish the construction of linear PCPPs by showing the following result.

Theorem V.3. *There exists an SLV with proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and rejection ratio $1/\operatorname{poly}\log n$.*

By combining the latter result with Theorem V.2, we obtain as a corollary the required linear PCPPs.

Corollary V.4. *There exists a linear PCPP with proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and rejection ratio $1/\operatorname{poly}\log n$.*

We first describe how to construct an SLV for the simple special case in which all the circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ are the same. Then, we show how to construct an SLV for a

more interesting case, which we call colorable constraint systems, by reducing it to few instances of the foregoing simple case. Finally, we show how to reduce the general case to the case of colorable constraint systems.

Intuition: As mentioned above, we begin by constructing an SLV for the case all the circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ are the same, which is relatively easy to do using error correcting codes. The main challenge lies in dealing with the general case, where the circuits may differ from one another.

One could hope to reduce the general case to the foregoing simple case as follows: Consider the universal circuit U, which takes as input a circuit φ and a string x, and outputs $\varphi(x)$. Now, instead of feeding the SLV with the circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$, we will feed the SLV with $\sqrt{n}$ copies of the universal circuit U, where the i-th copy is invoked on φ_i. This brings us back to the case where the SLV is invoked on $\sqrt{n}$ identical circuits, which we already know how to handle. A similar idea was used before in [7].

While the above idea is quite simple, when trying to implement it one runs into a variety of complications, with the main issue being that the universal circuit U itself is not a linear circuit. Thus, while the construction presented below is inspired by the above idea, its details are substantially different. Still, this idea of using the universal circuit is the core intuition that lies at the heart of this construction.

1) A simple case: We begin with describing how to construct an SLV V for case in which all the circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ are the same. Let $W \subseteq \{0,1\}^m$ denote the subspace that is accepted by the circuit $\varphi_1 = \varphi_2 = \ldots = \varphi_{\sqrt{n}}$.

In order to verify that $x_1, \ldots, x_{\sqrt{n}} \in W$, we consider a $\sqrt{n} \times m$ matrix M whose rows are exactly the vectors $x_1, \ldots, x_{\sqrt{n}}$. Let M' be the matrix obtained by encoding each *column* of M by some systematic linear code C with constant rate and relative distance. The proof string π of V is expected to contain all the rows of M' that do not belong to M.

Observe that if $x_1, \ldots, x_{\sqrt{n}} \in W$, then all the rows of M' belong to W, including the rows that are not in M. This is true since all the rows of M' are linear combinations of rows of M.

Now, when given oracle access to vectors $x_1, \ldots, x_{\sqrt{n}}$ and to a purported proof π, the verifier V acts as follows. The verifier V views its oracle as a purported matrix M', and performs the following checks:

1) V chooses a row of M' uniformly at random and checks that it belongs to W.
2) V chooses a column of M' uniformly at random and checks that it is a legal codeword of C.

V accepts if and only if both checks accept. It is easy to see that the query complexity and proof length of V are as required.

We turn to analyze the rejection ratio of V. Suppose that V is given oracle access to a purported matrix M', and that one of the rows of M' (say, the i-th row) is ε-far from W.

First, as a warm-up, assume that all the columns of M' are legal codewords of C. Observe that in this case, if at least one row of M' does not belong to W, then at least δ_C fraction of the rows of M' do not belong to W, where δ_C is the relative distance of C: To see it, suppose that the i-th row of M' does not belong to W, and let $w^\perp$ be any vector that is orthogonal to W but is not orthogonal to the i-th row of M'. Now, let $v = M' \cdot w^\perp$, and note that v is a non-zero codeword of C. Therefore, at least δ_C fraction of its coordinates are non-zero. However, for each non-zero coordinate of v, the corresponding row of M' is not orthogonal to $w^\perp$ and thus does not belong to W. Hence, at least δ_C fraction of the rows of M' do not belong to W, and V therefore rejects in this case with probability at least δ_C.

Next, assume that some of the columns of M' are not legal codewords of C, and let T denote the set of those columns. We may assume that T is of density at most ε or otherwise V rejects with probability at least ε. Consider the matrix M'_0 obtained by removing the columns in T from M, and the vector space W_0 obtained by projecting the vectors of W to the coordinates in $[m] \setminus T$. Now, if at least one row of M' is ε-far from W, then at least one row of M'_0 does not belong to W_0. We can now use the same argument as before to show that V rejects in this case with probability at least δ_C.

2) The case of colorable constraint systems: We proceed to show a construction of an SLV for circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ that are a colorable constraints system (CCS), to be defined below. We will later show that any sequence of circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ can be transformed to a CCS.

Informally, we say that a collection of circuits $\varphi_1, \ldots, \varphi_k : \{0,1\}^m \to \{0,1\}^t$ forms a CCS if the subspaces $W_{\varphi_1}, \ldots, W_{\varphi_k}$ can be described by a collection $\mathcal{S}$ of linear constraints that can be "legally colored" using few colors. We say that a coloring of the constraints in $\mathcal{S}$ is legal if constraints of the same color do not share coordinates. The idea that underlies our construction of an SLV for a CCS is that in a CCS can be decomposed to few monochromatic systems of constraints, such that each system can be reduced to the simple case discussed above. Some details follow.

For the following definitions and constructions, it will be convenient for us to describe linear subspaces by sets of linear constraints, defined next, rather than by linear circuits.

Notation V.5. Let $S \subseteq \{0,1\}^m$. We say that a subspace $W \subseteq \{0,1\}^m$ is the subspace described by S if W contains exactly the vectors that are orthogonal to all the vectors in S. We refer to the vectors of S as constraints. We say that a constraint $s \in S$ touches a coordinate $i \in [m]$ if $s_i = 1$.

Observe that every linear circuit $\varphi : \{0,1\}^m \to \{0,1\}^t$ can be transformed in polynomial time to a set S_φ of constraints which describes W_φ such that $|S_\varphi| = t$. This is

done simply by taking, for each output of φ, the constraint which touches exactly the coordinates that affect this output. We turn to define the notion of CSS.

Definition V.6 (Colorable constraints system)**.** Let $\chi \in \mathbb{N}$, let $\varphi_1, \ldots, \varphi_k : \{0,1\}^m \to \{0,1\}^t$ be linear circuits, and let $S_1, \ldots, S_k \subseteq \{0,1\}^m$ be the corresponding sets of constraints. We say that $\varphi_1, \ldots, \varphi_k$ form a χ-colorable constraints system (abbreivated χ-CCS) if the union $\mathcal{S} \stackrel{\text{def}}{=} S_1 \cup \ldots \cup S_k$ satisfies the following requirement: The constraints in $\mathcal{S}$ can be colored by χ colors, such that no two constraints in $\mathcal{S}$ of the same color touch the same coordinate.

A construction of SLV for a CCS: We turn to describe a construction of an SLV V for a CCS. Suppose that the verifier V is given as input circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ and is also given oracle access to vectors $x_1, \ldots, x_{\sqrt{n}} \in \{0,1\}^m$. Let $S_1, \ldots, S_{\sqrt{n}}$, and χ be as in Definition V.6.

Our strategy is to construct for each color $c \in [\chi]$ a collection of vectors $x_1^c, \ldots, x_{\sqrt{n}}^c \in \{0,1\}^m$ and a subspace $U^c \subseteq \{0,1\}^m$ such that the following holds: The circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ accept $x_1, \ldots, x_{\sqrt{n}}$ (respectively) if and only if the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ all belong to U^c *for every color* $c \in [\chi]$. The point is that verifying that $x_1^c, \ldots, x_{\sqrt{n}}^c$ belong to U^c can be done as in the simple case of Section V-C1. The proof will be expected to contain the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$, as well as additional information that allows verifying their consistency with $x_1, \ldots, x_{\sqrt{n}}$.

For each color $c \in [\chi]$, we define the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ to be the vectors obtained from $x_1, \ldots, x_{\sqrt{n}}$ by zeroing every coordinate that is not touched by a constraint of color c. More formally, for each x_j, we define $\left(x_j^c\right)_i \stackrel{\text{def}}{=} (x_j)_i$ if S_j contains a constraint of color c that touches the coordinate i, and $\left(x_j^c\right)_i \stackrel{\text{def}}{=} 0$ otherwise. We define the subspace $U^c \subseteq \{0,1\}^m$ to be the subspace that is described by all the constraints in $\mathcal{S}$ of color c. It is not hard to see that he circuits $\varphi_1, \ldots, \varphi_{\sqrt{n}}$ accept $x_1, \ldots, x_{\sqrt{n}}$ (respectively) if and only if the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ all belong to U^c *for every color* $c \in [\chi]$. Thus, if the proof indeed contains vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ that are constructed as defined above, we are done.

It remains to verify that the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ are obtained from $x_1, \ldots, x_{\sqrt{n}}$ as expected. To this end, observe that the vectors $x_1^c, \ldots, x_{\sqrt{n}}^c$ can be obtained from $x_1, \ldots, x_{\sqrt{n}}$ using multiplication. More specifically, for every $c \in [\chi]$ and $j \in [\sqrt{n}]$, we define a vector z_j^c by setting $\left(z_j^c\right)_i \stackrel{\text{def}}{=} 1$ if S_j contains a constraint of color c that touches the coordinate i, and $\left(x_j^c\right)_i \stackrel{\text{def}}{=} 0$ otherwise. We then observe that $x_j^c = x_j \cdot z_j^c$ (where the multiplication is *coordinate-wise*).

Let (C_A, C_B, C_M) be the multiplication codes of Assumption III.4. Now, the verifier checks the equality $x_j^c = x_j \cdot z_j^c$ by checking that the encodings $C_M(x_j^c)$ and $C_A(x_j) \cdot C_B(z_j^c)$ agree on a random coordinate. To this end, the verifier expects the proof to contain $C_A(x_j)$ and $C_M(x_j^c)$ (for every c and j), while $C_B(z_j^c)$ is computed by the verifier itself.

Finally, the verifier should verify that the purported vectors $C_A(x_j)$, $C_M(x_j^c)$ of the proof are indeed legal codewords of C_A and C_M, and this should be done for all j's simultaneously. However, this check can again be done as in the simple case of Section V-C1. This concludes our construction of the SLV.

Remark V.7. We note that the foregoing description slightly oversimplifies things. In particular, the encoding $C_M(x_j^c)$ is not well-defined because the message length of C_M is larger than $\left|x_j^c\right|$. Thus, in the actual construction, the proof is not required to contain $C_M(x_j^c)$ but is rather required to provide $C_A(x_j) \cdot C_B(z_j^c)$. We then use the fact that x_j^c is a prefix of the message encoded by $C_A(x_j) \cdot C_B(z_j^c)$, due to the fact that C_A, C_B, and C_M are systematic.

3) The general case: We conclude our construction of SLVs by showing that the general case can be reduced to the case of 6-CCS (i.e., CCS with 6 colors). The basic idea of the reduction is that every constraint system can be embedded on any routing network, and in particular one may choose a routing network whose edges can be colored using few colors. It should be mentioned that while the idea of embedding a constraint system on a routing network has been used in several prior works (e.g. [21], [22]), the use of this technique for constructing SLVs is new.

In order to embed a system of linear constraints on a routing network, we add auxiliary variables to the system in a way that essentially does not change the solution space. This notion of adding auxiliary variables while "essentially" not changing the solution space is captured by the following notion of *extension*.

Definition V.8. Let $W \subseteq \{0,1\}^m$ and let $l \in \mathbb{N}$. We say that a subspace $W' \subseteq \{0,1\}^{m+l}$ is an extension of W if it satisfies the following property: A vector $x \in \{0,1\}^m$ belongs to W if and only if there exists a vector $y \in \{0,1\}^l$ such that $x \circ y \in W'$.

In the foregoing Definition V.8, the vector y represents the assignment to the auxiliary variables, and the fact that the solution space remains intact is captured by the fact that the m long prefix of each vector in W' is in W, and by the fact that every vector in W has a corresponding vector in W'. Our reduction of a general constraint system to a CCS can now be stated as follows.

Claim V.9. *Let $\varphi_1, \ldots, \varphi_k : \{0,1\}^m \to \{0,1\}^t$ be linear circuits of size n. Then, one can transform $\varphi_1, \ldots, \varphi_k$ in polynomial time to circuits $\varphi'_1, \ldots, \varphi'_k : \{0,1\}^{m'} \to \{0,1\}^{t'}$ that form a 6-CCS, such that for each $i \in [k]$ the subspace*

$W_{\varphi'_i}$ is an extension of W_{φ_i}. Furthermore, the circuits $\varphi'_1, \ldots, \varphi'_k$ are of size $\tilde{O}(n)$, and it holds that $m' = \tilde{O}(n)$, $t' = \tilde{O}(n)$.

Proving Claim V.9 essentially finishes our work, since it is not hard to use it to construct SLVs for arbitrary linear circuits given this claim and the construction of Section V-C2. In the rest of this section we sketch the proof of Claim V.9. Fix a collection of circuits $\varphi_1, \ldots, \varphi_k$, let $W_1, \ldots, W_k$ be the subspaces accepted by those circuits, and let $S_1, \ldots, S_k$ be sets of linear constraints corresponding to $\varphi_1, \ldots, \varphi_k$. For each W_i, we denote by $W'_i \stackrel{\text{def}}{=} W_{\varphi'_i}$ the extension of W_i that we seek to construct.

Warm-up: As a warm-up, consider the case in which every set S_i is a collection of disjoint equality constraints. That is, we consider the case in which every constraint in S_i touches exactly two coordinates, and every coordinate is touched by exactly one constraint. We show that that such sets S_i can be extended to a 4-CCS - note that we use two colors less than in Claim V.9, a fact that will be used later.

We would like to construct for each W_i a subspace $W'_i \subseteq \{0,1\}^{m'}$ which is an extension of the subspace W_i, such that $W'_1, \ldots, W'_k$ form a 4-CCS. The idea that underlies the construction is the following. We identify every coordinate in $[m']$ with a vertex of a routing network G (see Section II-C). We then embed each equality constraint of the set S_i on a path in G that connects the coordinates touched by the constraint. The 4-colorability of the corresponding constraints system follows from the fact that G is 4-edge colorable. Some details follow.

Let G be a routing network with $m' = m \cdot \operatorname{poly}\log m$ vertices. We now construct a set S'_i of constraints that describes the subspace $W'_i \subseteq \{0,1\}^{m'}$ as follows. We begin by identifying each coordinate in $[m']$ with a vertex of G, and in particular identify the coordinates in $[m]$ with vertices of G. Next, we find a collection $\mathcal{P}$ of vertex-disjoint paths on G, such that for each equality constraint $s \in S_i$, there is a path in $\mathcal{P}$ that connects the coordinates that s touches. Finding such paths is possible by the fact that G is a routing network. Finally, for each edge e of G, we put in S'_i an equality constraint between the endpoints of e if and only if e participates in one of the aforementioned vertex-disjoint paths.

It should be clear that W'_i is an extension of W_i. To see that $W'_1, \ldots, W'_k$ form a 4-CCS, let $\mathcal{S}' \stackrel{\text{def}}{=} S'_1 \cup \ldots \cup S'_k$ and observe that every constraint in $\mathcal{S}'$ is an equality constraint between the endpoints of some edge of G. Now, we can choose the network G to be 4-edge colorable, in which case the constraints in $\mathcal{S}'$ can be colored using 4 colors such that no two constraints of the same color touch the same coordinate. Thus, $W'_1, \ldots, W'_k$ and $\mathcal{S}'$ satisfy Definition V.6 with $\chi = 4$. This concludes the construction.

Handling arbitrary linear circuits: It remains to show how to transform arbitrary linear circuits $\varphi_1, \ldots, \varphi_k$ to a 6-CCS as in Claim V.9. Below we sketch the basic idea of the transformation.

Given linear circuits $\varphi_1, \ldots, \varphi_k$, we consider the corresponding sets of constraints $S_1, \ldots, S_k$, and view them as systems of linear equations. Our goal is to simplify the systems such that they become a CCS, by adding auxiliary variables. The simplification is done by applying two operations:

1) Splitting a variable to several "copies", by adding new auxiliary variables and placing equality constraints between them and the original variable. The constraints that touched the original variable are split among its copies.
2) Permuting the variables, such that a constraint in S_i and a constraint inS_j ($i \neq j$) that previously touched different coordinates, now touch exactly the same coordinates, thus becoming the same constraint in $\mathcal{S} \stackrel{\text{def}}{=} S_1 \cup \ldots \cup S_k$.

The point is that the simplifcation operations may only add equality constraints, which we already know how to handle. After applying the simplication, we are left with a collection of *simple constraints* that may be colored using 2 colors, and with equality constraints that can be handled using 4 colors. We thus end up with a 6-CCS, as required.

VI. The PCP construction

In this section, we show how to prove our main theorem, restated below, by combining the tools that were developed throughout the extended abstract.

Theorem (I.1, main theorem, restated)**.** *For every time-constructible $t : \mathbb{N} \to \mathbb{N}$ and every language $L \in \mathbf{NTIME}(t)$, there exists a PCP verifier for L with proof length $\ell(n) = t(n) \cdot (\log(t(n)))^{O(\log\log t(n))}$, query complexity $O(1)$, and rejection probability $\Omega(1)$.*

Fix a time-constructible function t. We construct PCPs with the required parameters for $\mathbf{NTIME}(t)$. Our starting point is the linear PCPP constructed in Corollary V.4, which we denote here V_1. The verifier V_1 has proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and rejection ratio $1/\operatorname{poly}\log n$. Altough it is not shown in this work, it is not hard to modify V_1 such that it has $O(1)$-row/column access. We thus apply the robustization technique of Theorem IV.7 to V_1, resulting in a linear PCPP verifier V_2 that has proof length $\tilde{O}(n)$, query complexity $\tilde{O}(\sqrt{n})$, and *robustness* $1/\operatorname{poly}\log n$.

Our next step is to compose V_2 with itself for $\log\log n$ times, resulting in a linear PCPP verifier V_3 that has proof length $n \cdot (\log n)^{O(\log\log n)}$, query complexity $(\log n)^{O(\log\log n)}$, and rejection ratio $1/(\log n)^{O(\log\log n)}$. Then, we apply the transformation of linear PCPPs to (general) PCPs of Theorem III.6, resulting in a PCP verifier V_4 for $\mathbf{NTIME}(t)$, which has proof length $t \cdot (\log t)^{O(\log\log t)}$,

query complexity $(\log t)^{O(\log\log t)}$, and rejection probability $1/(\log t)^{O(\log\log t)}$.

We proceed by reducing the query complexity of V_4 to $O(1)$ by applying a standard query reduction technique that works roughly like the reduction of CIRCUITSAT to 3SAT. This technique increases the proof length by a factor that is polynomial in the original query complexity, and decreases the rejection probability by a similar factor. We are left with a PCP verifier V_5 for $\mathbf{NTIME}(t)$, which has proof length $t\cdot(\log t)^{O(\log\log t)}$, constant query complexity, and rejection probability $1/(\log t)^{O(\log\log t)}$.

Finally, we apply the gap amplification technique of Dinur [3] to V_5. This technique can be applied to PCPs with constant query complexity, and increases the rejection probability of a PCP verifier to a constant, while increasing the proof length by a factor that is inversly polynomial in the original rejection probability, and maintaining the constant query complexity. We therefore obtain a PCP verifier V_6 for $\mathbf{NTIME}(t)$, which has proof length $t\cdot(\log t)^{O(\log\log t)}$, constant query complexity, and constant rejection probability. V_6 is the required PCP verifier.

Remark VI.1. The foregoing description oversimplifies things a little. In particular, a few of the steps taken above require bounds on the randomness complexity and the decision complexity of the verifiers. However, such bounds can be proved.

ACKNOWLEDGEMENT

The author is grateful to Oded Goldreich for many useful discussions and ideas, and for comments that significantly improved the presentation of this work.

REFERENCES

[1] S. Arora and S. Safra, "Probabilistic checkable proofs: A new characterization of NP," *Journal of ACM volume*, vol. 45, no. 1, pp. 70–122, 1998, preliminary version in FOCS 1992.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and intractability of approximation problems," *Journal of ACM*, vol. 45, no. 3, pp. 501–555, 1998, preliminary version in FOCS 1992.

[3] I. Dinur, "The PCP theorem by gap amplification," *Journal of ACM*, vol. 54, no. 3, pp. 241–250, 2007, preliminary version in STOC 2006.

[4] O. Goldreich and S. Safra, "A combinatorial consistency lemma with application to proving the PCP theorem," *SIAM J. Comput.*, vol. 29, no. 4, pp. 1132–1154, 2000.

[5] I. Dinur and O. Reingold, "Assignment testers: Towards combinatorial proof of the PCP theorem," *SIAM Journal of Computing*, vol. 36, no. 4, pp. 155–164, 2006.

[6] I. Dinur and O. Meir, "Derandomized parallel repetition of structured PCPs," in *IEEE Conference on Computational Complexity*, 2010, pp. 16–27, full version can be obtained as ECCC TR10-107.

[7] O. Meir, "Combinatorial PCPs with efficient verifiers," in *FOCS*, 2009, full version is available as ECCC TR11-104.

[8] E. Ben-Sasson and M. Sudan, "Short PCPs with polylog query complexity," *SIAM J. Comput.*, vol. 38, no. 2, pp. 551–607, 2008, preliminary version in STOC 2005.

[9] O. Meir, "IP = PSPACE using error correcting codes," *Electronic Colloquium on Computational Complexity (ECCC)*, no. 137, 2010.

[10] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, "Robust PCPs of proximity, shorter PCPs and applications to coding," *SIAM Journal of Computing*, vol. 36, no. 4, pp. 120–134, 2006.

[11] E. Ben-Sasson, P. Harsha, O. Lachish, and A. Matsliah, "Sound 3-query PCPPs are long," *TOCT*, vol. 1, no. 2, 2009.

[12] E. Ben-Sasson and M. Sudan, "Robust locally testable codes and products of codes," *Random Struct. Algorithms*, vol. 28, no. 4, pp. 387–402, 2006, preliminary version in APPROX-RANDOM 2004.

[13] F. T. Leighton, *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

[14] L. G. Valiant, "Graph-theoretic arguments in low-level complexity," in *MFCS*, 1977, pp. 162–176.

[15] O. Goldreich and M. Sudan, "Locally testable codes and PCPs of almost linear length," *Journal of ACM*, vol. 53, no. 4, pp. 558–655, 2006, preliminary version in FOCS 2002, pages 13-22.

[16] M. Sudan, "Algorithmic introduction to coding theory (lecture notes)," 2001, available from http://theory.csail.mit.edu/~madhu/FT01/.

[17] I. Dinur, M. Sudan, and A. Wigderson, "Robust local testability of tensor products of ldpc codes," in *APPROX-RANDOM*, 2006, pp. 304–315.

[18] E. Ben-Sasson and M. Viderman, "Tensor products of weakly smooth codes are robust," *Theory of Computing*, vol. 5, no. 1, pp. 239–255, 2009.

[19] ——, "Composition of semi-ltcs by two-wise tensor products," in *APPROX-RANDOM*, 2009, pp. 378–391.

[20] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *J. Comput. Syst. Sci.*, vol. 43, no. 3, pp. 425–440, 1991.

[21] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *STOC*, 1991, pp. 21–31.

[22] A. Polishchuk and D. A. Spielman, "Nearly-linear size holographic proofs," in *STOC*, 1994, pp. 194–203.