Interactive proofs of proximity: Delegating computation in sublinear time

Interactive Proofs of Proximity: Delegating Computation in Sublinear Time

Guy N. Rothblum
Microsoft Research
rothblum@alum.mit.edu

Salil Vadhan*
Harvard University
salil@eecs.harvard.edu

Avi Wigderson†
Institute for Advanced Study
avi@ias.edu

ABSTRACT

We study interactive proofs with sublinear-time verifiers. These proof systems can be used to ensure approximate correctness for the results of computations delegated to an untrusted server. Following the literature on property testing, we seek proof systems where with high probability the verifier accepts every input in the language, and rejects every input that is *far* from the language. The verifier's query complexity (and computation complexity), as well as the communication, should all be sublinear. We call such a proof system an *Interactive Proof of Proximity* (IPP).

- On the positive side, our main result is that all languages in $\mathcal{NC}$ have Interactive Proofs of Proximity with roughly $\sqrt{n}$ query and communication and complexities, and polylog(n) communication rounds.

  This is achieved by identifying a natural language, membership in an affine subspace (for a structured class of subspaces), that is complete for constructing interactive proofs of proximity, and providing efficient protocols for it. In building an IPP for this complete language, we show a tradeoff between the query and communication complexity and the number of rounds. For example, we give a 2-round protocol with roughly $n^{3/4}$ queries and communication.

- On the negative side, we show that there exist natural languages in $\mathcal{NC}^1$, for which the sum of queries and communication in any constant-round interactive proof of proximity must be polynomially related to n. In particular, for any 2-round protocol, the sum of queries and communication must be at least $\tilde{\Omega}(\sqrt{n})$.

- Finally, we construct much better IPPs for specific functions, such as bipartiteness on random or well-mixing graphs, and the majority function. The query complexities of these protocols are provably better (by exponential or polynomial factors) than what is possible in the standard property testing model, i.e. without a prover.

Categories and Subject Descriptors

F.2.0 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*General*

Keywords

Interactive Proofs, Sublinear Algorithms

1. INTRODUCTION

The power of efficiently verifiable proof systems is a central question in the study of computation. In this work, we study the power, and the limitations, of interactive proof systems with very efficient *sublinear time* verifiers.

An interactive proof system [GMR89, BM88] is an interactive protocol between a prover and a randomized verifier, in which the prover convinces the verifier of the validity of a computational statement. The computational statement is usually the membership of an input x, known both to the prover and to the verifier, in a language L. The famous $\mathcal{IP} = \mathcal{PSPACE}$ Theorem [LFKN92, Sha92] established that interactive proofs with *polynomial-time* verifiers are remarkably powerful: they can be used to prove any language/statement computable in polynomial space. In this work, we follow the quest for efficient proof verification to its extreme. We ask:

Which computational statements have an interactive proof that can be verified in sublinear time?

Since the verifier runs in sublinear time, it cannot even read the input in its entirety. Following work on sublinear time algorithms and property testing [RS96, GGR98, EKR04], we consider verifiers that have query access to the input: the input x is treated as an oracle, and on query i the verifier receives $x[i]$. Our goal is sublinear verifier running time, but we also consider the *query complexity*, the *communication complexity*, and the *round complexity* of the protocol, as well as the running time of the honest prover: in different settings, queries, communication, and rounds of interaction may come at different costs. We emphasize that, as in the standard interactive proof model, *the verifier's input is reliable* and fixed throughout the protocol, whereas

*Center for Research on Computation and Society (CRCS) and School of Engineering and Applied Sciences (SEAS), Harvard University, Cambridge, MA USA. Work done in part while on leave as a Visiting Researcher at Microsoft Research SVC and a Visiting Scholar at Stanford University. Also supported by NSF grant CNS-1237235.

†Research partially supported by NSF grant CCF-0832797.

STOC'13, June 1-4, 2013, Palo Alto, California, USA.

the prover is unreliable and untrusted, and might deviate from the protocol in an adaptive and arbitrary manner. We find this to be the most natural model for sublinear-time proof verification, but we also provide some motivating applications below. Finally, throughout this work we consider computationally unbounded cheating provers, and make no cryptographic assumptions.

On reflection, it quickly becomes apparent that there exist simple and natural languages, for which membership cannot be verified in sublinear time. For example, there is no way for a prover to convince the verifier that the parity of a string is 0, unless the verifier reads the entire string. Thus, drawing further inspiration from property testing, we relax the requirement: the verifier should still accept inputs in the language, but it is only required to reject (with high probability) inputs that are *far from the language*, say at (fractional) Hamming distance at least $\varepsilon \in (0,1)$ from every input in the language.[1] We call such a protocol an *Interactive Proof of ε-Proximity*. We define and study a new complexity class: $\mathcal{IPP}$, the class of languages that have interactive proofs of proximity with sublinear-time verification. See Section 2 for formal definitions of these proof systems and the complexity class $\mathcal{IPP}$ (parameterized with the distance, query, communication and round complexities). For clarity, throughout the introduction we often ignore many of the proof system parameters. In particular, we refer loosely to the complexity class $\mathcal{IPP}$ as the class of languages that have Interactive Proofs of ε-Proximity, with sublinear query and communication complexities, for some non-trivial ε.

A Motivating Application: Delegating Computation in Sublinear Time. In addition to being a foundational issue in the theory of computation, proof verification is also motivated by real-world applications, such as delegating computation: a powerful server can run a computation for a weaker client, and provide an interactive proof of the output's correctness [GKR08]. The study of Interactive Proofs of Proximity with sublinear time verification is also motivated by such applications: namely, *delegating computation in sublinear time*. Consider a client that has reliable query access to an input x, but is very computationally restricted: it can only run in sublinear time. Still, this client wants to approximate a function f on x. A more powerful server can also access x, and can run in polynomial time, but it might cheat. The server can compute $y = f(x)$ and send it to the client. The client receives some (potentially incorrect) output y', and can use a sublinear time Interactive Proof of Proximity to verify that y' is "not too far" from $f(x)$. By "not too far" here, we mean that there exists some x' at Hamming distance ε from x s.t. $y' = f(x')$. Thus, even a sublinear time client can reliably delegate its computations, and verify the approximate accuracy of answers.

We emphasize again that we assume that the client's access to the input x is reliable: x is fixed in advance, or at the very least x does not change adaptively depending on the client's random coins. On the other hand, we assume nothing about the untrusted prover's behavior: it might cheat arbitrarily and adaptively, and it is not computationally bounded. The assumption that x is reliable warrants further discussion and motivation. For example, x might be a large dataset that both the client and the server can access (e.g. the Twitter graph). Alternatively, x might be real-time data that can be reliably measured by both the client and the server, e.g. network traffic or climate data. The client might only have the resources to measure (and store) x at a few points in time.[2] Finally, we note that even if the client's input is stored on an *untrusted* server, a memory checker [BEG+94] could be combined with an Interactive Proof of Proximity. The memory checker would detect any cheating on the client's queries to the input. Known constructions of memory checkers would maintain information-theoretic security, at only a sublinear communication overhead.

Comparison with Prior Work on Sublinear Time Computation and Verification. A rich body of work within the literature on sublinear-time algorithms focuses on *property testing* [RS96, GGR98]. There, a randomized tester has query access to the input, and needs to distinguish whether the input is in a language or far from the language. See also [Gol10b] and the excellent survey in [Ron09]. We extend this model by also providing interaction with a more powerful prover, who can read the input in its entirety, but might cheat. Of course, we assume more here than what is needed for standard property testing: namely, the existence of such a prover. However, we leverage this assumption to obtain more powerful and general results than what is possible in the property testing model. Most significantly, we tackle an inherent limitation of property testing: research on property testing has focused on constructing testers for languages based on their combinatorial or algebraic structure. This limitation seems inherent, because there exist simple and natural languages for which (provably) no sublinear time property testers exist (e.g. parity, see also [GKNR12]). In contrast, looking ahead, we show that IPPs allow general results for a rich class of computations. Further, we will show that even for specific problems, interaction with an untrusted prover can give a (provable) exponential improvement in query complexity compared to what is possible in property testing without a prover (even with only a single round of interaction and logarithmic communication).

Another beautiful line of research, starting with the work of Babai *et al.* on Holographic Proofs [BFLS91], has focused on "PCP-like" proof systems with sublinear-time verifiers. They constructed such proof systems under the assumption that the input is given in encoded form. Ergun, Kumar and Rubinfeld [EKR04] constructed PCP-like proofs for several natural languages. The work of Ben-Sasson *et al.* on PCPs of Proximity [BGH+06], and the work of Dinur and Reingold on Assignment Testers [DR06], give general constructions using PCP machinery. These works can be viewed as extending the property testing model by giving the verifier query access to a *fixed* proof string. While that proof string may be wrong, it is nonetheless fixed and does not change with verifier's queries to it. In our model (and in the delegating computation motivation) the prover can *adaptively* change its strategy and answers, as a function of subsequent verifier's messages. As discussed in [GKR08], handling adaptive cheating provers is essential for applications to delegating

[1] More generally, one could think of other distance measures or norms, as done in [EKR04]. We will (mostly) focus our attention on fractional Hamming distance throughout this work.

[2] In this scenario, the protocol might only be run later, after the real-time data has been observed. When this is the case, we may want an Interactive Proof where the verifier's queries can be made ahead of time, and are independent of the prover's messages. The verifiers in all of our protocols have this *"oblivious query"* property.

computation, and "PCP-like" proof systems are not directly suited for these applications. Further, this difference in the power of the cheating prover is driven home by the fact that in the standard setting, with polynomial time verifiers, the setting of a *fixed* proof string gives rise to the class $\mathcal{MIP}$ [BGKW88], and allows verification of languages in $\mathcal{NEXP}$ [BFL91], whereas the setting of an *adaptive* prover gives rise to the class $\mathcal{IP}$ [GMR89, BM88], and allows only verification of languages in $\mathcal{PSPACE}$ [LFKN92, Sha92]. Thus, we expect to be able to prove less than in the "PCP-like" setting, and indeed prove lower bounds to that effect.

The conference version of [EKR04] also studied interactive proofs with sublinear-time verifiers, and provided protocols for specific functions such as element distinctness.

1.1 The Power of IPP

In this section we outline our positive results. First, we show a general result, Theorem 1.1 below: every language computable by circuits of low depth,[3] has a sublinear-time Interactive Proof of Proximity. To prove this general result we identify a "complete problem" for constructing Interactive Proofs of Proximity: testing proximity to an affine subspace (form a structured class having to do with polynomial interpolation, see below). We provide an IPP for the complete problem, and then use completeness to derive IPPs for low-depth languages. In addition, we also mention our results for specific languages: IPPs for testing bipartiteness in the bounded degree model, and for testing Hamming weight. We find constructions for specific languages to be interesting even given the general result in Theorem 1.1. First, these constructions improve the communication and round complexities obtained there. The improvements can be quite dramatic: for bipartiteness we obtain a provable exponential improvement in the query complexity needed for IPPs compared with standard property testing (without a prover). Moreover, even if the communication or round complexities are not improved, construction for specific languages avoid the overhead induced by the completeness reduction.

IPPs for Low-Depth Computation. We show a general result for low-depth computations:

THEOREM 1.1 ($Depth(n^\gamma) \subseteq \mathcal{IPP}$). *For every language L computable by log-space uniform circuits of depth $D = D(n)$, size $S = S(n)$ and fan-in 2, and every distance $\varepsilon = \varepsilon(n)$,[4] there exists an Interactive Proof of ε-Proximity for L.*

The proof has perfect completeness and soundness error 1/2. For an input of length n, the query complexity is $(1/\varepsilon)^{1+o(1)}$, the communication complexity is $(\varepsilon \cdot n \cdot (1/\varepsilon)^{o(1)} \cdot \mathrm{poly}(D))$, and the number of rounds is $O(\log n + D \cdot \log S)$. The honest Prover $\mathcal{P}$ runs in time $\mathrm{poly}(S)$, and the verifier $\mathcal{V}$ runs in time $O(((1/\varepsilon) + (\varepsilon \cdot n))^{1+o(1)} \cdot \mathrm{poly}(D) \cdot \log S)$.

See Section 3 and the full version for further details. We note that Theorem 1.1 gives sublinear-copmlexity IPPs even for very small values of ε,[5] e.g. for $\varepsilon = 1/n^{1-\delta}$ for any constant $\delta > 0$. In interpreting this result, one useful setting to consider is that of an $\mathcal{NC}$ language (computable in depth $D(n) = \mathrm{polylog}(n)$ and size $S(n) = \mathrm{poly}(n)$). Here Theorem 1.1 gives a tradeoff between the query complexity q and communication complexity c, where $q \times c = n^{1+o(1)}$. The number of rounds is poly-logarithmic, and the honest prover runs in polynomial time. In particular, for any $\varepsilon(n) \geq n^{-1/2}$, the queries, communication, and verifier runtime can all be be $n^{1/2+o(1)}$. This applies to varied problems in $\mathcal{NC}$: for example, graph functions such as planarity testing, connectivity, finding a perfect matching, or algebraic problems such as computing matrix rank..

REMARK 1.2 (NEARLY-OPTIMAL QUERY COMPLEXITY). *In general, query complexity $\Omega(1/\varepsilon)$ is essential for IPPs: a cheating prover can always choose $x \in L$ and flip $(\varepsilon \cdot n)$ uniformly random coordinates. Suppose that w.h.p this gives x' that is ε-far from L (e.g. L is an error-correcting code, so no two strings in L are close). When running the protocol on x', the cheating prover can follow the honest prover's strategy for input x, and unless the verifier queries one of the coordinates on which x and x' differ it will accept (by completeness). Since x and x' differ only on $(\varepsilon \cdot n)$ uniformly random coordinates, the verifier must make $\Omega(1/\varepsilon)$ queries in order to reject x' (as needed for soundness).*

A "Complete" Problem: Testing Membership in an Affine Subspace. For input length n, an integer $t \ll n$, and a finite field F, an affine subspace is specified by a matrix $A \in F^{t \times n}$ and a "target" vector $v \in F^t$. The language *AffineMem*(A, v), of checking membership in the affine subspace defined by A and v, is the set of inputs $x \in F^n$ for which $A \cdot x = v$.[6] An IPP for *AffineMem* allows a verifier to check whether its input $x \in F^n$ is close to a vector $x' \in F^n$ such that $A \cdot x' = v$, using only a sublinear (in n) number of queries to x, and sublinear communication. Looking ahead, we will focus on testing membership in a specific structured family of affine subspaces. In particular, the matrix A will have a succinct representation. See further details below.

We provide a reduction, via an interactive protocol, from proving proximity to any language L computable in low depth, to proving proximity to *AffineMem* (for a restricted class of matrices A). This reduction uses the "Interactive Proofs for Muggles" of [GKR08] (see Section 3).

THEOREM 1.3 (INFORMAL; *AffineMem* IS "COMPLETE"). *For every distance bound $\varepsilon = \varepsilon(n)$, and every language L computable by log-space uniform boolean circuits of depth $D = D(n)$, size $S = S(n)$, and fan-in 2, for $t = t(n) = O(\varepsilon \cdot n \cdot \log n)$ and a finite field F, there exists an interactive protocol between a prover and a verifier as follows. On input x, the output is (an implicit description of) a matrix $A \in F^{t \times n}$ and an (explicit) vector $v \in F^t$, such that:*

- *Completeness. If $x \in L$ and the prover honestly follows the protocol, then $x \in$ AffineMem(A, v).*
- *Soundness. If x is ε-far from L, then for any cheating prover, with all but 1/2 probability over the verifier's coins, x is also ε-far from AffineMem(A, v).*

[3] Throughout, when we say a language is computable in "low depth" or $Depth(n^\gamma)$, we mean that it can be computed by a log-space uniform family of Boolean circuit with fan-in 2 and depth $O(n^\gamma)$, for a universal constant $\gamma > 0$.

[4] Here and throughout, we assume that all complexity and distance parameters such as $D(n)$, $S(n)$, $\varepsilon(n)$ are computable in space $O(\log n)$.

[5] The smaller ε is, the "harder" it is to prove ε-proximity. In particular an Interactive Proof of ε-Proximity, is also an interactive proof of ε'-proximity for any $\varepsilon' \geq \varepsilon$.

[6] Or rather, the language is specified by an ensemble of matrices and target vectors, one for each input length.

The prover runs in time $\text{poly}(S)$*, and the verifier runs in time* $\varepsilon \cdot n \cdot \text{poly}(D)$*. The communication complexity is* $\varepsilon \cdot n \cdot \text{poly}(D)$*, and the number of rounds is* $O(D \cdot \log S)$*. The verifier makes no queries to the input* x *during the protocol.*

As noted above, we focus our attention on a structured variant of *AffineMem*, the problem *PVAL* of evaluating (or interpolating) bounded-degree multivariate polynomials. In fact, the formal statement of Theorem 1.3 shows completeness for this structured variant, and we use the additional structure to directly construct an IPP for *PVAL*. This, in turn, yields (via completeness) an IPP for any low-depth language, and in particular also for the more general *AffineMem* problem (where the matrix A has any implicit representation computable in low depth).

The problem *PVAL* is defined as follows. Each input x specifies a multivariate bounded-degree polynomial P_x over a field F. An affine subspace is now defined by a size-t set $J \subseteq F^m$ of "evaluation points", and a "target" vector $v \in F^t$ of values. The language $PVAL(J, v)$ is the set of inputs x s.t. $\forall j \in J, P_x(j) = v[j]$ (where $v[j]$ denotes entry of v corresponding to item j).[7] One difference from the more general *AffineMem* problem, is that here the (large) matrix A (of size $t \cdot n$) is implicitly specified by the set J (of size t).

In more detail: the input x specifies the polynomial P_x as a list of its evaluations on a canonical subset $S \subseteq F^m$ (we call these the "interpolation points"). E.g., for polynomials of degree less than k in each variable, we can take $S = H^m$ for any fixed $H \subseteq F$ of size k (usually F will be an extension field of H). Taking $n = k^m$, the input x gives an evaluation for each interpolation point, yielding a unique polynomial P_x of degree less than k in each variable that agrees with these evaluations. For any point $j \in F^m$ and a "target value" $v[j] \in F$, the constraint $P_x(j) = v[j]$ is a linear constraint over P_x's evaluations on the interpolation points (using the linearity of polynomial interpolation). I.e., we can express this constraint as $A_j \cdot x = v[j]$, where $\mathcal{A}_j \in F^n$ is a (fixed) function of j. The language $PVAL(J, v)$ is exactly $\mathit{AffineMem}(A_J, v)$, where for each $j \in J$ the matrix A_J includes the row A_j defined above.

An IPP for *PVAL*. The goal of a sublinear verifier for *PVAL*, is to determine whether the given evaluations of the polynomial on evaluation points in J are correct, while only reading a sublinear number of interpolation points. We note that the polynomial's evaluation on each point in J might well depend on most (if not all) of the evaluations on interpolation points. Thus, the verifier cannot (on its own) verify any of the claimed evaluations on J without reading $\Omega(n)$ coordinates of x. Moreover, for natural choices of J and v the distribution generated by drawing a uniformly random input $x \leftarrow PVAL(J, v)$ can be $\Omega(n)$-wise independent: In particular, for any $n/10$ queries made by the verifier into the input, the values it sees are uniformly random. A uniformly random input, however, is (1/10)-far from $PVAL(J, v)$ with all but negligible probability. We are asking the verifier to distinguish these two distributions using a sublinear number of queries! Indeed, this is impossible in the standard property testing model (without a prover), because the verifier should accept a random input in the language, and reject a uniformly random input, but its views of the input in both cases are statistically close.

In the presence of a prover, however, we can overcome these difficulties, and construct an Interactive Proof of Proximity for *PVAL*.

THEOREM 1.4 (INFORMAL; IPP FOR *PVAL*). *For every* $n \in \mathbb{N}$ *and* $\varepsilon > 0$*, for* J, v, F, k, m *as generated in the reduction of Theorem 1.3, there exists an* ε*-IPP for* $PVAL(J, v)$*. The IPP has perfect completeness, soundness* $1/2$*, query complexity* $(1/\varepsilon)^{1+o(1)}$*, and* $O(\log(1/\varepsilon))$ *rounds. The communication complexity is* $\varepsilon \cdot n \cdot (1/\varepsilon)^{o(1)}$*, and the verifier runtime is* $((1/\varepsilon) + \varepsilon \cdot n)^{1+o(1)}$*. The honest prover runtime is* $\text{poly}(n)$*.*

More generally, we get a tradeoff between the communication complexity and round complexity. The theorem statement above is one extreme of this tradeoff (optimized for small communication). We can also optimize for the round complexity, and obtain a 3-message protocol with $\sqrt{\varepsilon} \cdot n \cdot (1/\varepsilon)^{o(1)}$ communication. See Corollaries 3.2 and 3.3 in Section 3 for formal statements.

REMARK 1.5. *We emphasize that the IPP setting is different from the setting of local testing or decoding, and from the setting of PCP-like proofs, in that a malicious prover can be completely adaptive in its cheating. Thus, it is not obvious how to use the strong local testing and decoding properties of polynomials. In particular, one initial approach might be to have the prover perform polynomial evaluations for the verifier, and use local testing/decoding to detect cheating, as is done throughout the PCP literature and (in particular) in the work of [BGH+06]. The difficulty with this approach, is that local testing and decoding procedures are only sound against non-adaptive (static) corruptions, and are easily fooled by adaptive corruptions.*

A Taste of Techniques. We proceed with intuition for the proof of Theorem 1.4. Consider the case where the input x specifies a 2-variate polynomial $P_x(i_1, i_2)$ of degree less than $k = \sqrt{n}$ in each variable. x includes the evaluations of P_x on all the specification points H^2, where $H \subseteq F$ is of size $\sqrt{n}$. We can "break up" $P_x(i_1, i_2)$ into H univariate polynomials $\{Q_{i_1}(i_2) = P_x(i_1, i_2)\}_{i_1 \in H}$. Each univariate Q_{i_1} is specified by $\sqrt{n}$ bits of the input. Moreover, for any $j = (j_1, j_2) \in F$, we can use polynomial interpolation to express $P_x(j)$ as a linear combination of $\{Q_{i_1}(j_2)\}_{i_1 \in H}$, where the coefficients in the linear combination depend only on j_1.

With this view in mind, consider an evaluation point $j = (j_1, j_2) \in J$. The prover claims that $P_x(j) = v[j]$. In the IPP, for each $j \in J$, the prover sends $\sqrt{n}$ (alleged) values $\{Q_{i_1}(j_2)\}_{i_1 \in H}$. The verifier checks (via polynomial interpolation, as above), that these values are consistent with the "j-th claim" $P_x(j) = v[j]$. If the j-th claim was false, and the values are consistent with it, then there must be some $i_1 \in H$ for which the prover sent a false value v_{i_1, j_2} for $Q_{i_1}(j_2)$. We have gone from $|J|$ claims about the "large polynomial" P_x (specified by n variables), to $|J| \cdot \sqrt{n}$ claims about $\sqrt{n}$ "small polynomials" $\{Q_{i_1}\}_{i_1 \in H}$. While each small claim depends only on $\sqrt{n}$ input bits, it is not clear whether we have made any progress: verifying all of the new claims still requires reading every bit of the input x.

Suppose, however, that the prover has been careless in its cheating: in particular, for each $j = (j_1, j_2) \in J$, the prover

[7] To be more precise, *PVAL* is an infinite ensemble of finite languages indexed by the field F, the number of variables, the degree bound, the subset J of points, and the vector v of values on J.

lies about the value of $Q_{i_1}(j_2)$ for a different $i_1 \in H$. In this case, the verifier can pick a uniformly random $i_1 \in H$, and check all $|J|$ claims on Q_{i_1}. Observe that this only requires reading $\sqrt{n}$ input bits of x: the evaluations of P_x on points in $\{i_1\} \times H$. Since we assumed (with loss of generality) that for $|J|$ of the choices of i_1 there exists some claim on Q_{i_1} where the prover is cheating, the verifier will catch the prover with probability $|J|/|H|$. This would already imply an IPP with sublinear complexity (for the appropriate choice of $|J|$). The main challenge is that a cheating prover need not be so careless—in particular, the prover may focus all of its false claims on a single small polynomial Q_{i_1}. In this case, the verifier cannot detect the cheating by testing randomly chosen Q's. This is the main technical challenge for constructing a non-trivial IPP. Our first observation is that, in this case, Q_{i_1} is "far from" the prover's claims: the $\sqrt{n}$ input bits that specify Q_{i_1} are very far from any $\sqrt{n}$ bits that specify a polynomial Q'_{i_1} that agrees with the prover's claims. In the IPP, the verifier combines the prover's claims about the different Q's by taking a random linear combination of them $\hat{Q} = \sum_{i_1 \in H} \alpha_{i_1} \cdot Q_{i_1}$. This induces $|J|$ new claims about the values of $\hat{Q}$ at $|J|$ points in F. We show that if there is a Q_{i_1} that is "far from" the prover's claims, then, with high probability over the linear combination, the new polynomial $\hat{Q}$ will be "far from" the induced claims. This is shown using a new lemma about distances between linear subspaces, which may be of independent interest:

LEMMA 1.6. *Let S and T be two linear subspaces of F^n (for any finite field F and $n \in \mathbb{N}$). Suppose that there exists some point $s \in S$ such that s is at (fractional) Hamming distance at least ε from T (i.e., at Hamming distance at least ε from every $t \in T$).*

Then, with all but $1/(|F|-1)$ probability over the choice of a uniformly random point $r \in_R S$, the distance between r and T is at least $\varepsilon/2$.

Omitting many non-trivial details, the final step in the IPP is for the verifier to check consistency of the induced claims about $\hat{Q}$ viz a viz the input x. We show that this can be done using sublinear query and communication complexities. Note that the above was all for the case of 2-variate polynomials, and required only a constant number of rounds. In the full protocol we can handle m-variate polynomials by composing a protocol based on these ideas to repeatedly reduce the number of variables (using $O(m)$ rounds of communication). We conclude the technical overview by remarking that the protocol for $PVAL$ can be generalized to a natural problem of evaluating tensor powers of a linear code. We take this more general view throughout Section 3 and in the full version.In this we follow, and are inspired by, the work of Meir [Mei10], who illuminated the central role of tensor codes in the construction of standard interactive proofs. Here too, tensors of linear codes play an essential role in constructing interactive proofs of proximity. See Section 3 and the full version for further details.

1.1.1 IPPs for Specific Languages

We now move to specific languages, for which we provide tailored IPPs that are more efficient (and simple) than the general construction of Theorem 1.1.

An IPP for Graph Bipartiteness. We construct an Interactive Proof of proximity for bipartiteness on random or well-mixing graphs in the bounded degree model (see e.g. [Gol10a]). The protocol accepts every bipartite graph. For a random graph of bounded degree, or for any graph that is both far from bipartite and well mixing, the verifier will reject with high probability. That is, this is a protocol for a promise problem, where YES instances are bipartite, and NO instances are far from bipartite and well-mixing (thus a random bounded-degree graph will be a NO instance with high probability). For constant distance parameter $\varepsilon = \Theta(1)$, the protocol has $O(\log n)$ query and communication complexities, and only a single round of communication. This is a particularly sharp illustration of the benefits of interaction with a prover, as Goldreich and Ron [GR02] proved a $\Omega(\sqrt{n})$ lower bound on the query complexity of any property tester for this problem (i.e. without the help of a prover). The IPP uses ideas from Goldreich and Ron [GR99], but leverages the presence of a prover to obtain (provably) an exponential improvement in the query complexity (using only a single round of interaction, and logarithmic communication). An interesting open problem is extending this IPP to apply to all graphs (not just rapidly-mixing ones), as was done by [GR99] in the standard property testing model (they build a $\tilde{O}(\sqrt{n})$-query tester for all graphs). See the full version for details.

An IPP for Majority/Hamming Weight. We also consider the language *Hamming*(w), of vectors in $\{0,1\}^n$ with Hamming weight w. We give a direct Interactive Proof of ε-Proximity for this problem, with query and communication complexities $\tilde{O}(1/\varepsilon)$, and logarithmically many rounds. This protocol can be used by a sublinear-time verifier, with the help of a more powerful prover, to approximate the Hamming weight of a given vector to within distance ε. In particular, it can also be used to prove that the (approximate) majority of bits in a boolean vector are 1 (where the approximation is up to fractional additive error ε). Here too, there is a provable improvement on what can be obtained without a prover: by sampling lower bounds, approximating the Hamming weight to within additive fractional error ε requires $\Omega(1/\varepsilon^2)$ queries. We note that, similarly to remark 1.2, $\Omega(1/\varepsilon)$ queries are necessary, and so this result is essentially tight. See the full version for details.

1.2 The Limits of IPP

We prove lower bounds on the tradeoffs between query complexity, communication complexity, and round complexity, showing that constant-round interactive proofs of proximity must have query or communication complexity polynomially related to n. Specifically, we exhibit a uniform $\mathcal{NC}^1$ language $L_n \subseteq \{0,1\}^n$ (in fact an $\mathbb{F}_2$-linear subspace) for which any $O(1)$-round interactive proof of $(1/8)$-proximity with communication complexity $c(n)$ and query complexity $q(n)$ must have $\max\{c(n), q(n)\} = n^{\Omega(1/r(n))}$. So achieving polynomially small complexity (as we do) is the best we can hope for in bounded-round protocols. (It remains an intriguing open problem whether using many rounds, polylogarithmic communication and query complexity is possible.)

To prove our lower bound, we first use standard transformations of interactive proofs [GS86, BM88] to convert any bounded-round, private-coin IPP into an "AM-type" IPP with related complexity. By an "AM-type" IPP, we mean a 2-message public-coin protocol, where the verifier sends its coin tosses to the prover, receives a message from the prover, and then deterministically queries its input oracle and de-

cides whether to accept or reject. For AM-type protocols, we obtain a lower bound of $\max\{c(n), q(n)\} = \tilde{\Omega}(\sqrt{n})$ by using a novel connection with pseudorandomness for CNF formulas. Specifically, we show that the maximum acceptance probability of the verifier in an AM-type IPP protocol can be computed as the probability that the input x satisfies a random CNF formula of size at most $2^{c(n)+q(n)}$ chosen according to the following distribution: choose the coins v of the verifier at random, and then consider the CNF formula that takes an OR over the $2^{c(n)}$ possible prover messages p and checks whether the input x yields any of the $\leq 2^{q(n)}$ accepting sequences of responses to the verifier's oracle queries. Thus, we can take our language L_n to be the image of a pseudorandom generator for CNF formulas of size $2^{c(n)+q(n)}$ with seed length $O((c(n)+q(n))^2 \log n)$, e.g. as constructed in [Nis91, Baz09]. The protocol will have roughly the same acceptance probability on a random element of L_n as it does on a uniformly random string, which will be far from L_n with high probability unless $\max\{c(n), q(n)\} = \tilde{\Omega}(\sqrt{n})$ (since otherwise L_n has exponentially small support). See Section 4 for further details.

1.3 Further Remarks and Related Work

Related Work on Interactive Proofs. Interactive Proofs were introduced by Goldwasser, Micali and Rackoff [GMR89], and independently by Babai and Moran [BM88]. A long line of research has considered various restrictions on the verifier in an interactive proof (see the full version), but the running time of the verifiers in these works is at least linear in the input length.

In terms of the running time of the *honest* prover, we follow [GKR08] in focusing on polynomial-time honest provers, and on interactive proofs for tractable languages. All of our protocols for specific languages have efficient honest provers. The general result of Theorem 1.1 applies to every language that is computable in low-depth (even with super-polynomial size circuits), but we emphasize that the honest prover running time in that protocol is polynomial in the circuit size. If the circuit is polynomial size, then the honest prover runs in polynomial time.

Computationally Sound Proof Systems. We note that *under cryptographic assumptions*, if we only require security against polynomial-time cheating provers, then even more powerful general results are possible (see below). In contrast, all of our protocols are unconditionally secure, and make no assumptions about the power of a cheating prover. We further remark that our negative results show a provable separation between what can be obtained by unconditionally sound versus computationally sound protocols (under cryptographic assumptions).

We briefly elaborate on computationally sound interactive proofs of proximity. These can be obtained by combining PCPs of Proximity (or Assignment Testers) [BGH$^+$06, DR06], with the techniques of Kilian and Micali [Kil88, Mic94] for ensuring non-adaptive behavior of the (polynomial-time) cheating prover. The idea is for the prover to construct a PCP of Proximity, and then Merkle-hash it down to a short string that is sent to the verifier. The Merkle hash forces non-adaptive behavior later, when the prover answers the verifier's queries. These techniques require assuming a family of collision-resistant hash functions (CRHs). The end result is a 2-round computationally sound interactive proof of ε-proximity for any language in $\mathcal{EXP}$. The prover's running time is polynomial in the running time needed to decide the language, the verifier's running time is polylogarithmic in this running time (and polynomial in the security parameter). The query complexity is $(1/\varepsilon) \cdot \text{polylog}(n)$, and the communication complexity is polylogarithmic in n and polynomial in the security parameter. Micali's construction can be used to further reduce the round complexity to a single round in the random oracle model.

2. DEFINITIONS AND PRELIMINARIES

Notation. For a finite field F and a vector $\vec{x} \in \Sigma^n$, we use $|\vec{x}|$ to denote the length of $\vec{x}$. We use $\vec{x}[i]$ to refer to the i-th element of $\vec{x}$. For a subset $I \subseteq [n]$, $\vec{x}_{|I} \in F^{|I|}$ is the projection of $\vec{x}$ onto coordinates in I. The (absolute) Hamming weight $\|\vec{x}\|$ is the number of non-zero coordinates in $\vec{x}$. For two vectors $\vec{x}, \vec{y}$ of the same length, the (fractional) Hamming distance between $\vec{x}, \vec{y} \in F^n$, which we denote $\Delta(\vec{x}, \vec{y})$, is the fraction of coordinates on which $\vec{x}$ and $\vec{y}$ disagree ($\|\vec{x} - \vec{y}\|/n$). We sometimes refer to the *absolute* Hamming distance $\Delta_{absolute}(\vec{x}, \vec{y}) = \Delta(\vec{x}, \vec{y}) \cdot n$. Throughout this work, when we refer to Hamming distance we mean *fractional* distance unless we explicitly note otherwise.

For a matrix $X \in F^{k \times \ell}$, and $(i, j) \in [k] \times [\ell]$, we use $X[i][j]$ to refer to the item in the i-th row and j-th column of X. We use $X[i, *] \in F^{\ell}$ to refer to the i-th row, and $X[*, j] \in F^k$ to refer to the j-th column. For a subset of coordinates $I \subseteq [k] \times [\ell]$, we use $X_{|I}$ to denote the *vector* of X's values in coordinates in I (in lexicographic order). We define $|X|$, $\|X\|$, and the Hamming distance between matrices analogously to vectors.

In several places throughout this work, we view vectors as matrices. For $\vec{x} \in F^n$, we specify k and ℓ such that $n = k \cdot \ell$, and associate every coordinate in $[n]$ with a pair $(i, j) \in [k] \times [\ell]$ in the natural way. We view $\vec{x}$ as a matrix $X \in F^{k \times \ell}$, where every element of $\vec{x}$ is mapped to the appropriate coordinate in X.

Interactive Proofs of Proximity. These are protocols used for verifying that an input is *close to* a language L. The goal is achieving this with a verifier that runs in sublinear time, without even reading the input in its entirety. The input to the proof system is shared by the prover and the verifier. For greater generality, as in [BGH$^+$06], we divide the input into an implicit or oracle part x, and an explicit part w. The prover gets (x, w) as a (standard) input. The verifier gets $n = |x|$ ($O(\log n)$ bits) and w as a standard input, and gets oracle access to x. For a language L over pairs (x, w), we refer to the language $L(w) = \{x : (x, w) \in L\}$. For example, the $PVAL$ language has an explicit input $(F, k, m, J, \vec{v})$, where F is a finite field, k an individual degree bound, m a number of variables, J a set of evaluation coordinates, and $\vec{v}$ a vector of values. The implicit input x is the description of a multivariate polynomial p_x. An input $(x, (F, k, m, J, \vec{v}))$ is in $PVAL$ if p_x and $\vec{v}$ agree on the evaluation points in J. Alternatively, x is in $PVAL(F, k, m, J, \vec{v})$ iff $(x, (F, k, m, J, \vec{v})) \in PVAL$.

DEFINITION 2.1 (INTERACTIVE PROOF OF PROXIMITY (IPP)). *For a language L over alphabet $\Sigma = \Sigma(n)$, and distance $\varepsilon = \varepsilon(n)$, a proof system $(\mathcal{P}, \mathcal{V})$ is an* Interactive Proof of ε-Proximity *for a language L on pairs (x, w), with $\alpha = \alpha(n)$ and $\beta = \beta(n)$ completeness and soundness errors (respectively), if the following conditions hold for all x of length n and strings w,*

- *α-Completeness: if $(x, w) \in L$, then with all but α probability over the coins of $\mathcal{V}$, it will accept in the protocol execution $(\mathcal{P}(x, w), \mathcal{V}^x(n, w))$.*

- *β-Soundness: if x is at fractional Hamming distance ε or more from L, then for every cheating prover $\mathcal{P}'$, with all but β probability over the coins of $\mathcal{V}$, it will reject in the protocol execution $(\mathcal{P}'(x, w), \mathcal{V}^x(n, w))$*

We call x the implicit input*, and w the* explicit input *(which may be empty). A* public-coin protocol *is one in which the verifier's messages are simply random coin tosses. A protocol with completeness error 0 is said to have* perfect completeness*. The* query complexity $q = q(n)$ *is the number of queries $\mathcal{V}$ makes into x (each query returns an element in the alphabet Σ). The* communication complexity $c = c(n)$ *is the number of bits exchanged. The* round complexity $r = r(n)$ *is the number of communication rounds (with two messages per round).*

Definition 2.2 ($\mathcal{IPP}$ Complexity Class). *The class $\mathcal{IPP}(\varepsilon, q, c, r, v, p)$ includes all languages that have Interactive Proofs of $\varepsilon(n)$-proximity, with query, communication, and round complexities $q(n)$, $c(n)$, and $r(n)$ (respectively), where the verifier running time is $v(n)$ and the honest prover's running time is $p(n)$.*

For ease of notation, we also define $\mathcal{IPP}(\varepsilon, q, c, r)$, the class of languages as above, where the verifier's running time is $\text{poly}(q(n), c(n), \log n)$ and the prover's running time is $\text{poly}(n)$. Finally, on occasion we refer to $\mathcal{IPP}(\varepsilon)$, the class of languages as above, where the query and communication complexities, and the verifier's running time, are all sublinear in n, and the honest prover's running time is $\text{poly}(n)$.

3. MORE ON THE POWER OF IPP

In this section we describe our positive results on Interactive Proofs of Proximity for Low-Depth Languages. These are obtained by first identifying the complete language *PVAL*: we give a reduction from proving proximity to any language computable in low depth, to proving proximity to *PVAL*. Our main technical contribution is an Interactive Proof of Proximity for *PVAL*. By completeness, this protocol immediately implies an Interactive Proof of Proximity for every language computable in low depth (see Theorem 1.1 in the introduction).

As discussed in the introduction, the implicit input X to *PVAL* specifies an m-variate polynomial over a finite field F. The language is specified by a set of points and values, and a polynomial X is in the language if it agrees with the given values on the given points. Equivalently, we can think of *PVAL* as the language of evaluating coordinates in a Reed-Muller encoding of the input X (a message for the code). A message X is in the language if its encoding, projected onto the given coordinates, agrees with the given values.

Definition 3.1 (*PVAL*). *PVAL is a language on pairs: X and $(F, k, m, J, \vec{v})$, where X is the implicit input, and the other inputs are explicit. $X \in F^{k^m}$ is the description m-variate polynomial p_X of degree at most k in each variable. Taking $[k]$ to be the lexicographically first k elements of F, X specifies p_X by its evaluations on all points in $[k]^m$ (we refer to $[k]^m$ as the "interpolation set"). $J \subseteq F^m$ is a set of evaluation points, and $\vec{v} \in F^{|J|}$ a vector of values. The input is in PVAL if and only if the polynomial p_X takes the values specified by $\vec{v}$ on the coordinates in J.*

Equivalently, taking C^m to be an m-variate Reed-Muller code over F (with degree parameter k in each variable), the input is in PVAL if and only if the encoding $C^m(X)$ takes the values specified by $\vec{v}$ on the coordinates in J: $C^m(X)_{|J} = \vec{v}$.

We say that $(X, J, \vec{v})$ is an instance of PVAL on (F, k, m) or on C^m, if and only if it is the case that $(X, (F, k, m, J, \vec{v})) \in PVAL$. We say that $(X, J, \vec{v})$ is d-far from PVAL (on (F, k, m) or on C^m) if and only if X is d-far from $PVAL(F, k, m, J, \vec{v})$).

We begin with an overview of our contributions: the completeness reduction to *PVAL*, and the Interactive Proof of Proximity for *PVAL*, highlighting the main technical contributions along the way. The full details follow.

Overview: *PVAL* is "Complete". We show a reduction from proving proximity to any language L computable in (log-space uniform) size $S = S(n)$ and depth $D = D(n) = n^\gamma$, to proving proximity to *PVAL*. This uses the "Interactive Proofs for Muggles" of [GKR08]. They showed an interactive proof for any such L, where the prover runs in time $\text{poly}(S)$, and the round complexity, communication complexity, and verifier running time are all polynomial in D. That Interactive Proof, when run on an input $X \in \{0,1\}^n$, outputs a claim about an encoding of X at a uniformly random coordinate j. The encoding is performed using an m-variate Reed-Muller code C^m. The protocol's output is a pair (j, v), specifying the claim $C^m(X)[j] = v$. Completeness means that, when $X \in L$ and the prover honestly follows the protocol, the claim specified by (j, v) is indeed true. Soundness means that, for $X \notin L$, for any cheating prover strategy, with probability at least $1/2$ over the verifier's coins, the claim specified by (j, v) is false, i.e. $C^m(X)[j] \neq v$.

Our reduction runs the GKR interactive proof t times in parallel. This yields t pairs of the form (j, v), which we aggregate into a collection J of coordinates in $C^m(X)$, and a vector $\vec{v}$ of claimed values for those coordinates. Completeness means that, when $X \in L$ and the prover honestly follows the protocol, the claim specified by $(J, \vec{v})$ is true, i.e. $C^m(X)_{|J} = \vec{v}$. Soundness means that when $X \notin L$, for any cheating prover strategy, w.h.p. the claim specified by $(J, \vec{v})$ is false, i.e. $C^m(X)_{|J} \neq \vec{v}$. On closer inspection, soundness actually implies a stronger property: If X is at (fractional) Hamming distance ε from L, where $t \geq 2\varepsilon \cdot n \cdot \log n$ from L, then w.h.p. over the verifier's coins, $C^m(X')_{|J} \neq \vec{v}$ *for all X' at distance less than ε from X (simultaneously).* To see this, observe that every X' at distance less than ε from X is also not in L (because X is ε-far from L). By soundness of each invocation of the GKR protocol, with probability at least $1/2$, the claim on X' specified by that invocation's output is false, i.e. $C^m(X')[j] \neq v$. The probability that, over t independent parallel runs of the GKR protocol, $C^m(X')_{|J} = \vec{v}$ is at most $2^{-t} \leq n^{-2\varepsilon \cdot n}$. The number of inputs at (fractional) distance less than ε from X is at most $n^{\varepsilon \cdot n}$. Taking a Union Bound over all inputs X' at distance less than ε from X, we get that w.h.p. there exists no such X' for which the claim specified by $(J, \vec{v})$ is true.

We conclude that there is a reduction from any language $L \in Depth(n^\gamma)$ to *PVAL* on the code C^m used by the GKR protocol. We note that the reduction specifies an instance X for $PVAL(J, \vec{v})$, where $|J| = O(\varepsilon \cdot n \cdot \log n)$. In our use of the GKR verifier, it sees J and $\vec{v}$, but never accesses

the input X.[8] The completeness of $PVAL$ motivates our main contribution: an Interactive Proof of Proximity for this problem (and through it, for any language in $Depth(n^\gamma)$).

Overview: Interactive Proof of Proximity for $PVAL$. As discussed above, the difficulty in proving proximity to $PVAL$, is that each coordinate in the encoding $C^m(X)$ may well depend on most (if not all) of X's coordinates. The verifier cannot (on his own) verify any of the prover's claims about coordinates in $C(X)$ without reading $\Omega(n)$ coordinates of X. Indeed, we do note know how to directly build an Interactive Proof of Proximity for verifying the values of an *arbitrary* code C^m in a specified set of coordinates.[9]

Luckily, C^m is not an arbitrary code: it is a Reed-Muller code, which (among its many useful properties) is a Linear Tensor Code (see the full version for background on Tensor codes). This is the main structural property of $PVAL$ that we will exploit. More generally, we will define the language $TVAL$ for evaluating any tensor code C^m of a linear code C. Analogously to $PVAL$, $TVAL$ (on C^m) is specified by a set of codeword coordinates J and values $\vec{v}$. A message X is in $TVAL$ if $C^m(X)_{|J} = \vec{v}$ (see the full version for details).

Our main result in this section is an Interactive Proof of Proximity for $TVAL$ on any tensor of a linear code. In particular, this immediately gives an interactive proof for $PVAL$ (which is $TVAL$ on a Reed-Muller code). The full theorem statement and proof are in the full version. This theorem gives a tradeoff between the number of messages exchanged and the communication complexity. Rather than give the more general statement, we focus for now on the two corollaries outlined in the introduction. We consider the case $|J| = O(\varepsilon \cdot n \cdot \log n)$ (as in instances produced by the reduction from $Depth(n^\gamma)$ to $PVAL$).

COROLLARY 3.2. *For any $n \in \mathbb{N}$ and $\varepsilon \in (0,1]$, take $k = \sqrt{1/\varepsilon}$ and $m = \log_k n$. For any linear code $C : F^k \to F^\ell$, where $\ell = \text{poly}(k)$, $|F| = \text{polylog}(n)$. There exists an Interactive Proof of ε-Proximity for $TVAL$ on C^m, with perfect completeness, soundness $1/2$, query complexity $(\varepsilon)^{1+o(1)}$ and 3 messages. For $|J| = O(\varepsilon \cdot n \cdot \log n)$, the communication complexity is $(n \cdot \sqrt{\varepsilon}) \cdot (1/\varepsilon)^{o(1)}$, and the verifier runtime is $((1/\varepsilon) + (n \cdot \sqrt{\varepsilon}))^{1+o(1)}$. The honest prover runtime is* $\text{poly}(n)$.

COROLLARY 3.3. *For any $n \in \mathbb{N}$ and $\varepsilon \in (0,1]$, take $k = \log n$ and $m = \log_k n$. For any linear code $C : F^k \to F^\ell$, where $\ell = \text{poly}(k)$, $|F| = \text{polylog}(n)$, there exists an Interactive Proof of ε-Proximity for $TVAL$ on C^m, with perfect completeness, soundness $1/2$, query complexity $(1/\varepsilon)^{1+o(1)}$, and $O(\log(1/\varepsilon)/\log\log n)$ messages. For $|J| = O(\varepsilon \cdot n \cdot \log n)$, the communication complexity is $\varepsilon \cdot n \cdot (1/\varepsilon)^{o(1)}$, and the verifier runtime is $((1/\varepsilon) + (\varepsilon \cdot n))^{1+o(1)}$. The honest prover runtime is* $\text{poly}(n)$.

Protocol Intuition. We highlight some of the main ideas from this protocol, see also the discussion in the introduction (further details are in the full version). We begin with some high-level background. Let F be a finite field. Let $C : F^k \to F^\ell$ be a linear code (where $\ell = \text{poly}(k)$). The (linear) Tensor Code $C^2 : F^{k^2} \to F^{\ell^2}$ is obtained by treating the input message as a matrix $X \in F^{k \times k}$, encoding each of X's k rows using the base code C, and then encoding each of the ℓ resulting columns using C. This yields a codeword $C^2(X) \in F^{\ell \times \ell}$, which we also view as a matrix. We will use the code C^2 for our high level overview here. More generally, the m-th tensor C^m is obtained by treating the input message as a matrix $X \in F^{k \times k^{m-1}}$, encoding each of X's k rows using the $(m-1)$-th tensor C^{m-1}, and then encoding each of the resulting ℓ^{m-1} columns using C (see the full version for details).

The Interactive Proof of Proximity, on input X, should accept if $C^2(X)_{|J} = \vec{v}$, and reject if X is at Hamming distance at least ε from every X' s.t. $C^2(X')_{|J} = \vec{v}$. For this informal exposition, we take $\varepsilon = k^{1/2}/n$ and $|J| = t = O(\varepsilon \cdot n \cdot \log n) = \tilde{O}(k^{1/2})$. We use $d = \varepsilon \cdot n$ to denote the *absolute* Hamming distance. We aim for $o(k^2)$ query and communication complexities (sublinear in the input size k^2).

Viewing $C^2(X)$ as an $\ell \times \ell$ matrix, for each $j \in J$, the (honest) prover sends to the verifier the entire column of $C^2(X)$ that contains coordinate j. In fact, it suffices to send the first k coordinates of each such column, as the remaining column coordinates are simply the encoding of the first k coordinates using the code C. Let $Y \in F^{k \times |J|}$ be the resulting matrix (as computed by an honest prover). The verifier receives from the prover some $Y' \in F^{k \times |J|}$ (possibly the prover is cheating and $Y' \neq Y$), and checks that Y' is consistent with the values claimed in $\vec{v}$; namely that for each $j \in J$, the encoding of the first k coordinates in the column of Y' that contains coordinate j, is consistent with the value that $\vec{v}$ gives for coordinate j.

For completeness, when $Y = Y'$ and $X \in TVAL(J, \vec{v})$, the verifier will accept in the above checks. When X is ε-far from $TVAL(J, \vec{v})$, however, the prover must "cheat" in many of Y''s columns in order to avoid detection. Observe that the i-th row of Y contains the encoding of the i-th row of X in a coordinate set $J[1]$.[10] Thus, from Y' we get k "new" claims about the encodings of X's rows in a subset of coordinates. Each of these claims is an instance of $TVAL$ on the base code C: Using $X[i,*]$ and $Y'[i,*]$ to denote the i-th rows of X and Y' (respectively), for each $i \in [k]$ we get a new instance $(X[i,*], J[1], Y'[i,*])$ of $TVAL$ on C.

When the original instance X is not in $TVAL(J, \vec{v})$ on C^2, at least one of the k new instances will not be in $TVAL$ on C (because Y' is consistent with the values claimed in $\vec{v}$). Let d_i denote the (absolute) distance of the i-th new instance from $TVAL$ on C. Since the original instance is at absolute distance $d = \varepsilon \cdot n = k^{1/2}$ from $TVAL$ on C^2, we get that $\sum_{i \in [k]} d_i \geq d = k^{1/2}$. We want the verifier to efficiently test the k new instances, in query complexity that is smaller than the query complexity needed to test the original instance. One immediate challenge is that it might be the case that *all but one* of the instances are "kosher" (i.e. in $TVAL$), and the prover's cheating is concentrated in a single instance specified by some row $i^* \in [k]$. The verifier doesn't know i^*, and so it must test the validity of all k of the new claims. We deal with this challenge using new ideas described below.

[8] In its original setting, the GKR verifier runs an additional test to check that the prover's claim about the input encoding in true. This requires (quasi)-linear time, and so we avoid this step.

[9] We can use our general results to obtain an Interactive Proof of Proximity to verify the values of any code C whose encoding can be computed in low depth.

[10] For each $j \in J$, we view it as a pair $(j[0], j[1]) \in \ell \times \ell$, and $J[1]$ contains the coordinate $j[1]$.

When The Prover Cheats on a Single Row. First, assume that the prover's cheating is concentrated in a single row i^* with $d_{i^*} = d$ (we discuss the more general case later). The verifier sends to the prover coefficients specifying a random linear combination $\vec{z} \in F^k$ of X's rows. The (honest) prover sends back $\vec{x} = \vec{z} \cdot X \in F^k$. The verifier receives some $\vec{x}'$, and verifies $C(\vec{x}')_{J^{-(0)}} = \vec{y}' = \vec{z} \cdot Y'$ (otherwise the verifier rejects). This additional test maintains perfect completeness, because C is a linear code. For soundness, we consider the case where every other row $i \neq i^*$ of Y' contains the correct values of the i-th row of X. W.h.p. row i^* will have a non-zero coefficient in the linear combination. When this is the case, we get that $(\vec{x}, J^{-(0)}, \vec{y}')$ is also at absolute distance $d_{i^*} = d$ from $TVAL$. Thus the absolute Hamming distance between the "real" $\vec{x}$, and $\vec{x}'$ sent by the cheating prover, is at least d (since $(\vec{x}', J^{-(0)}, \vec{y}') \in TVAL$).

To guarantee soundness, the verifier now checks a few randomly chosen coordinates of $\vec{x}'$, to see that it is close to the "real" $\vec{x}$. If the prover is cheating, then the (absolute) Hamming distance between $\vec{x}$ and $\vec{x}'$ is at least d, so it suffices to check $O(k/d) = O(k^{1/2})$ coordinates of $\vec{x}'$. Each coordinate in $\vec{x}$ is a linear combination of k coordinates in X, and so the total query complexity is $O(k^{3/2})$. The communication complexity is $\tilde{O}(k^{3/2})$ for sending Y' (sending $\vec{z}$ and $\vec{x}'$ takes only $O(k)$ bits of communication).

When The Prover Cheats on Many Rows. More generally, the prover's cheating might be spread over many (or all) rows. We show that there always exists a set I of rows, such that $\forall i \in I, d_i = \tilde{\Omega}(d/|I|)$. We saw above, that when the cheating is all on one row ($|I|$=1), then w.h.p. a random linear combination that contains this row specifies an instance that is at distance $\Omega(d)$ from $TVAL$ on C. In fact, this is but an example of a more general (and more powerful) phenomenon; We show that, with high probability, a random linear combination $\vec{z}$ of the rows, which contains one or more of the rows in the set I, specifies an instance of distance $\tilde{\Omega}(d/|I|)$. This result follows from a new lemma about distances between linear subspaces, which may be of independent interest (see Lemma 1.6 in the introduction).

With this new insight in mind, suppose that the verifier knows $|I|$ (but doesn't know which rows are in I). The verifier takes a random linear combination of $O(k/|I|)$ rows. W.h.p. the linear combination will include at least one row of I, and by the above it will specify an instance of distance $\tilde{\Omega}(d/|I|)$. There is a tradeoff here: the larger I is, the smaller the distance of the resulting instance, but *the "locality" of $\vec{x}$ is reduced*: each bit of $\vec{x}$ depends on only $O(k/|I|)$ bits of the original input X. When the cheating prover sends $\vec{x}'$, we now know that $\Delta_{absolute}(\vec{x}, \vec{x}') = \tilde{\Omega}(d/|I|)$. After receiving $\vec{x}'$, the verifier can test $\tilde{O}(k/(d/|I|)) = \tilde{O}(|I| \cdot k/d)$ random coordinates, and w.h.p the prover's cheating will be caught. Testing each coordinate requires $O(k/|I|)$ queries to the input X, and so the total query complexity remains $\tilde{O}(k^2/d) = \tilde{O}(k^{3/2})$, regardless of $|I|$.

The above assumed that the verifier knows $|I|$. In reality, the verifier doesn't know what $|I|$ is, but it can (approximately) cover all possibilities by running $\log k$ instantiations of the protocol, and trying all possible powers of 2. This blows up the query and communication complexities by a $\log k$ factor, giving a complete and sound sublinear protocol for $TVAL$ on Tensor codes C^2. The query and communication complexities are $\tilde{O}(k^{3/2}) = \tilde{O}(n^{3/4})$.

Composing Tensor Reductions. More generally, the above protocol for $TVAL$ can be used to provide a reduction from testing $TVAL$ on a linear tensor code $C_1 \otimes C_2$, to testing ($\log k$ instances of) $TVAL$ on the C_2. We thus call it the "Tensor Reduction" protocol. (see the full version)

Starting with an instance of $TVAL$ for a high-power Tensor code C^m, we can use $C^m = C \otimes C^{m-1}$, and repeatedly compose the Tensor Reduction Protocol, reducing the tensor power more and more. This improves the communication complexity (but increases the number of rounds).

4. LOWER BOUNDS

In this section, we prove lower bounds on the tradeoff between query complexity, communication complexity, and round complexity for interactive proofs of proximity:

THEOREM 4.1. *There is a language L in logspace-uniform $\mathcal{NC}^1$ such that in any interactive proof of (1/8)-proximity for L with prover-to-verifier communication $c(n)$, verifier query complexity $q(n)$, and $r(n)$ verifier messages (where the verifier messages are uniformly random strings, and verifier has no additional randomization), we have*

$$\min\{r(n), c(n), q(n)\} = n^{\Omega(1/r(n))}.$$

In particular, it is impossible to have $r(n) = O(1)$ and $c(n) = q(n) = n^{o(1)}$.

Moreover, for every length n, $L \cap \{0,1\}^n$ is an $\mathbb{F}_2$ subspace of $\{0,1\}^n$ for which a basis can be constructed in logspace-uniform $\mathcal{NC}$.

For lack of space, the proofs are omitted. See the full version. We begin by proving a lower bound for an "$\mathcal{AM}$ analogue of $\mathcal{IPP}$" (which we denote $\mathcal{AM}$-$\mathcal{IPP}$), where on input length n, the verifier sends its random coins $v \stackrel{R}{\leftarrow} \{0,1\}^{t(n)}$ to the prover, the prover replies with a message $p \in \{0,1\}^{c(n)}$, and then the verifier makes $q(n)$ queries to the input oracle $x \in \{0,1\}^n$ (depending on n, v and p) and then accepts or rejects (based on v, p, and the oracle responses).

Our lower bound is based on the observation that the maximum acceptance probability of the verifier in such a protocol can be computed by a "small" CNF formula.

LEMMA 4.2. *Let V be a verifier in an $\mathcal{AM}$-$\mathcal{IPP}$ protocol where on inputs of length n, the prover's message has length $c(n)$ and the verifier makes $q(n)$ queries to its input oracle $x \in \{0,1\}^n$. Let $\mathrm{Acc}_V(x)$ denote the maximum, over all prover strategies P^*, that the verifier V accepts on input x. Then for every input length n, there is a distribution $\mathcal{D}$ on CNF formulas with at most $2^{c(n)+q(n)}$ clauses of width at most $q(n)$ such that $\mathrm{Acc}_V(x) = \mathrm{E}_{\varphi \stackrel{R}{\leftarrow} \mathcal{D}}[\varphi(x)]$.*

Given that the verifier's acceptance probability is computed by (distributions over) CNF formulas, we can use a pseudorandom generator for CNF formulas to obtain a language for which it is hard to prove proximity.[11]

LEMMA 4.3. *Let V be a verifier in an interactive protocol, and $\mathrm{Acc}_V(x)$ be the maximum probability, over all prover strategies P^*, that the V accepts on input x. Suppose G_n :*

[11]This has the flavor of results in proof complexity, where for certain proof systems it is hard to certify that a given string is outside the image of an appropriate PRG [ABSRW04].

$\{0,1\}^{n/8} \to \{0,1\}^n$ *be a generator that fools* Acc_V *in the sense that*

$$\mathrm{E}[\mathrm{Acc}_V(G_n(U_{n/8}))] - \mathrm{E}[\mathrm{Acc}_V(U_n)] < 1/8.$$

Then V cannot be a verifier in an interactive proof of $n/8$-proximity for $\mathrm{Image}(G_n)$.

We can now use known explicit pseudorandom generators that fool CNF formulas, such as [Nis91, Baz09], to obtain languages that have no low-complexity $\mathcal{AM}$-$\mathcal{IPP}$ protocol.

LEMMA 4.4. *For every n, there is an explicit generator $G_n : \{0,1\}^{n/8} \to \{0,1\}^n$ such that*

1. *For every CNF formula φ on n variables with at most $m(n)$ clauses, for $m(n) = 2^{-\Omega(\sqrt{n/\log n})}$, we have:*

$$|\mathrm{E}[\mathrm{Acc}_V(G_n(U_{n/8}))] - \mathrm{E}[\mathrm{Acc}_V(U_n)]| < 2^{-\Omega(\sqrt{n/\log n})}$$

2. *G_n is an $\mathbb{F}_2$-linear map, and the $n \times n/8$ matrix for evaluating G_n can be constructed in logspace-uniform $\mathcal{NC}$.*

THEOREM 4.5. *There is a language L in logspace-uniform $\mathcal{NC}$ that has no $\mathcal{AM}$-$\mathcal{IPP}$ proof of $(1/8)$-proximity in which the prover's message has length $o(\sqrt{n/\log n})$ and the verifier's query complexity is $o(\sqrt{n/\log n})$. Moreover, for every length n, $L \cap \{0,1\}^n$ is an $\mathbb{F}_2$ subspace of $\{0,1\}^n$ for which a basis can be constructed in logspace-uniform $\mathcal{NC}$.*

Now, to handle protocols with more rounds and private coins, we use classic transformations to convert the protocols to $\mathcal{AM}$ protocols, keeping track of the effect of the transformations on the communication complexity and the query complexity. The first transformation collapses any r-round public-coin interactive proof to an $\mathcal{AM}$ proof system, with a complexity blow-up that is exponential in r. This is a more quantitative version of the $\mathcal{AM}$ Collapse Theorem of Babai and Moran [BM88], as worked out in [GVW02]. See the full version for details.

5. REFERENCES

[ABSRW04] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Pseudorandom generators in propositional proof complexity. *SIAM J. Comput.*, 34(1):67–88, 2004.

[Baz09] Louay M. J. Bazzi. Polylogarithmic independence can fool dnf formulas. *SIAM J. Comput.*, 38(6):2220–2272, 2009.

[BEG+94] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 21–31, 1991.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[BGKW88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 113–131, 1988.

[BM88] László Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the pcp theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006.

[EKR04] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate probabilistically checkable proofs. *Inf. Comput.*, 189(2):135–159, 2004.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[GKNR12] Oded Goldreich, Michael Krivelevich, Ilan Newman, and Eyal Rozenberg. Hierarchy theorems for property testing. *Computational Complexity*, 21(1):129–192, 2012.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[Gol10a] Oded Goldreich. Introduction to testing graph properties. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:82, 2010.

[Gol10b] Oded Goldreich, editor. *Property Testing*, volume 6390 of *LNCS*. Springer, 2010.

[GR99] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[GR02] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *STOC*, pages 59–68, 1986.

[GVW02] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1-2):1–53, 2002.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[Mei10] Or Meir. Ip = pspace using error correcting codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:137, 2010.

[Mic94] Silvio Micali. Cs proofs (extended abstract). In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 436–453, 1994.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

[Ron09] Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.