Proceedings of the International Congress of Mathematicians
Hyderabad, India, 2010

Probabilistically Checkable Proofs and Codes

Irit Dinur *

Abstract. **NP** is the complexity class of problems for which it is easy to check that a solution is correct. In contrast, finding solutions to certain **NP** problems is widely believed to be hard. The canonical example is the SAT problem: given a Boolean formula, it is notoriously difficult to come up with a satisfying assignment, whereas given a proposed assignment it is trivial to plug in the values and verify its correctness. Such an assignment is an "**NP**-proof" for the satisfiability of the formula.

Although the verification is simple, it is not *local*, i.e., a verifier must typically read (almost) the entire proof in order to reach the right decision. In contrast, the landmark PCP theorem [4, 3] says that proofs can be encoded into a special "PCP" format, that allows speedy verification. In the new format it is guaranteed that a PCP proof of a false statement will have many many errors. Thus such proofs can be verified by a randomized procedure that is *local*: it reads only a *constant* (!) number of bits from the proof and with high probability detects an error if one exists.

How are these PCP encodings constructed? First, we describe the related and possibly cleaner problem of constructing *locally testable codes*. These are essentially error correcting codes that are testable by a randomized local algorithm. We point out some connections between local testing and questions about stability of various mathematical systems. We then sketch two known ways of constructing PCPs.

Mathematics Subject Classification (2000). 68Q17

Keywords. Probabilistically Checkable Proofs, PCP, Locally Testable Codes.

1. Introduction

In this paper we discuss the computational complexity class **NP** and a robust characterization of this class through Probabilistically Checkable Proofs (PCPs). We describe the

NP is the complexity class of problems for which it is easy to check that a solution is correct. In contrast, finding solutions to **NP** problems is widely believed to be hard. Consider for example the problem 3-SAT[1]. Given a 3-CNF Boolean formula, it is notoriously difficult to come up with a satisfying assignment, whereas given a proposed assignment it is trivial to plug in the values and verify its correctness. Such an assignment is an "**NP**-proof" for the satisfiability of the formula. Indeed, an alternative way to define **NP** is as the class of all sets $L \subset \{0,1\}^*$ that

*Weizmann Institute of Science
[1]3-SAT is defined in Section 2.

have efficient proof systems: proof systems in which there is a polynomial-time algorithm that verifies correctness of the statement $x \in L$ with assistance of a proof. This significantly generalizes systems such as Frege's propositional calculus in which a proof system is defined by a set of axioms and inference rules, and a valid proof consists of a sequence of steps that are either axioms or inferred from previous steps through an inference rule.

Intuitively, a proof is very sensitive to error. A false theorem can be "proven" by a proof that consists of only one erroneous step. Similarly, a 3-SAT formula φ can be unsatisfiable, yet have an assignment that satisfies all clauses save one. In these cases, the verifier must check *every single proof step / clause* in order to make sure that the proof is correct.

Probabilistically Checkable Proofs. In contrast, the PCP theorem gives each set in **NP** an alternative proof system, in which proofs are robust. In this system a proof for a false statement is guaranteed to have so many errors that a verifier can randomly read only a few bits from the proof and decide, with high probability of success, whether the proof is valid or not.

More formally, a PCP verifier for a set $L \in \mathbf{NP}$ is an extension of the standard NP verifier. Whereas the standard verifier is given an input $x \stackrel{?}{\in} L$ and access to a proof π and is required to accept or reject, the PCP verifier is also allowed to read some r random bits. However, it is restricted to read only at most q bits from the proof. The class $PCP[r, q]$ is defined to contain all languages L for which there is a verifier V that uses $O(r)$ random bits, reads $O(q)$ bits from the proof, and guarantees the following. Let $V^\pi(x, \rho)$ denote the output of V on input x, randomness ρ, and proof π.

- (Completeness:) If $x \in L$ then there is a proof π such that
$$\Pr_\rho[V^\pi(x, \rho) \text{ accepts}] = 1.$$
- (Soundness:) If $x \notin L$ then for any proof π,
$$\Pr_\rho[V^\pi(x, \rho) \text{ accepts}] \leq \frac{1}{2}.$$

The PCP theorem says that every $L \in \mathbf{NP}$ has a verifier that uses at most $O(\log n)$ random bits and reads only $O(1)$ bits from the proof. In other words,

Theorem 1.1 (PCP Theorem, [4, 3]). NP $\subseteq$ PCP$[\log n, 1]$.

Consider, as an example, the PCP verifier for 3-SAT. Given an instance, i.e. a 3-CNF Boolean formula φ, the PCP verifier reads φ, but is only allowed access to a *constant* q number of bits from a proof string π. What should be written in these bits? This clearly cannot be the "obvious" proof which is just an assignment to the variables of φ. Such a proof will miserably fail the soundness condition: unsatisfiable formulae that can be almost satisfied will fool the verifier into accepting with too high a probability.

Locally Testable Codes. The fact that in a PCP proof system, a proof for a false statement is guaranteed to have many errors begs the analogy to error correcting codes. An error correcting code is a mapping $C : \{0,1\}^k \to \{0,1\}^n$ in which even a tiny distance between two message strings $x \neq y \in \{0,1\}^k$ is guaranteed to become huge: the encoded strings $C(x), C(y)$ will differ on at least (say) 20% of their bits. In this analogy, it is the number of erroneous steps in a proof that needs to be greatly amplified.

Can a PCP proof be constructed simply by encoding the standard NP proof? This seems like a promising path to follow since error correcting codes are easy to come by. In fact, the answer is yes, but with a caveat. A simple error correcting code will certainly not do the trick . The encoding must be much more subtle, and the main additional ingredient that is needed is *local testability*. Local testability is the ability to decide if a string is a valid codeword by looking at a (randomly selected) small part of it. This leads us to the definition of *locally testable codes (LTCs)* whose construction is the combinatorial heart of constructing PCPs. LTCs are a clean mathematical analog of PCPs whose definition is direct and requires no mention of computation. We will discuss testability and LTCs in Section 4.

Stability. Local testability is related to the notion of stability of mathematical systems. Generally speaking, a system of constraints (e.g. equations) is considered stable if small perturbations of the system result in small perturbations of the solution set. In such systems the only way to approximately satisfy the system is by taking a valid solution and perturbing it. In other words, approximate solutions are always perturbations of exact solutions.

In the discrete setting, when the constraints are, for example, Boolean, a system of constraints is *stable* if any approximate solution, i.e. one that satisfies many of the constraints, must be close to a perfect solution, i.e. one that satisfies the entire system. This notion naturally appears in various other mathematical contexts, and there are interesting connections between results on PCPs and stability results in areas such as discrete Fourier analysis, geometry, probability, and arithmetic combinatorics.

Stability and hardness of approximation. The PCP machinery allows one to transform any system (that belongs to **NP**) into a stable system. The transformation can be done efficiently, even if solving the system is infeasible. For example, the following is equivalent to the PCP theorem:

Theorem 1.2 (Informal Statement). *There is an efficiently computable transformation r_{PCP} that on input a 3-CNF formula φ generates a 3-CNF formula $\varphi' = r_{\text{PCP}}(\varphi)$ such that the set $s(\varphi') = \{x \mid x \text{ satisfies } \varphi'\}$ is stable (i.e., whenever x' satisfies many of the clauses of φ' it must be close to some $x \in s(\varphi')$).*

Feige et. al. [16] were the first to discover the equivalence of this theorem and Theorem 1.1, and this has had far reaching implications for the complexity of approximation problems. We will discuss this further in Section 3.2.

Let us point out that Theorem 1.2 implies that if $\mathbf{P} \neq \mathbf{NP}$ there is no algorithm that inputs a 3-CNF formula and approximates the maximal number of satisfiable clauses to within increasingly better precision. The reason is that by applying such an algorithm on $r_{\text{PCP}}(\varphi)$ one can determine if φ is satisfiable or not, thus solving an **NP**-complete problem.

In other words, we have just established the hardness of finding, even approximately, the maximal number of satisfiable clauses in a 3-SAT formula.

Gap amplification. The key to constructing PCPs is a transformation that amplifies errors in a proof, had there been any in the first place. The original proof and formulation of the PCP theorem stemmed out of research on proof verification. The techniques used in the proof are largely based on algebraic encodings and testing results that are generally called "low degree tests". More recently, a combinatorial proof was given by the author [13]. This proof is described more naturally as a hardness of approximation result, and it relies on rapid mixing of random walks on expanding graphs. In Section 5 we sketch these two approaches.

Organization. We begin in Section 2 with basic definitions, as well as an introduction of the class **NP** aimed at the non-experts. In Section 3 we formally state the PCP theorem, and connect it to hardness of approximation problems. In Section 4 we discuss stability and local testable codes, and give a concrete example of a locally testable code. This is intended to provide some intuition as to how PCPs work. Finally, in Section 5 we sketch the algebraic and combinatorial constructions of PCPs.

2. Preliminaries

2.1. Computational Problems. A computational (search) problem is formally described by a relation $S \subset \{0,1\}^* \times \{0,1\}^*$. We interpret the pair $(x, y) \in S$ to mean that y is a valid solution for problem instance x. Some examples are

- SATISFY $= \{(\varphi, a)\}$ where φ describes a Boolean logic formula; and a describes a satisfying assignment to the variables (i.e., an assignment under which the formula evaluates to true).
- CLIQUE $= \{(G, K)\}$ where G describes a graph, and K describes a clique in the graph, i.e. a set of vertices every pair of which are connected by an edge.
- PROOFS $= \{(T, \pi)\}$ where T describes a theorem in some fixed logic proof system, and π describes a proof for T in that proof system.

An optimization problem is a search problem S together with a valuation function $v : \{0,1\}^* \to \mathbb{R}^+$ that assigns a value to each solution. For example the maximum clique problem is the search problem CLIQUE together with a valuation

function that counts the number of vertices in a given solution. The goal is to find a clique of largest size.

An algorithm is called *efficient* if its running time is bounded by a polynomial function in the length of the input. Whenever we consider an algorithm it is implicitly assumed to be efficient. An algorithm solves an optimization (maximization or minimization) problem if for every instance $x \in \{0,1\}^*$ it finds a $y \in \{0,1\}^*$ such that $(x,y) \in S$ (if one exists) and $v(y)$ is optimal (maximal or minimal).

An *r-approximation* algorithm for a given combinatorial optimization problem is an algorithm that always finds a solution whose value is within multiplicative factor r of the optimal value.

It is often simpler to work with decision problems. A decision problem, or a language, is a set $L \subset \{0,1\}^*$. For a search problem S, denote by $S(x) = \{y \mid (x,y) \in S\}$. A set L is a called a *decision version* of a search problem S if

$$L = \{x \in \{0,1\}^* \mid S(x) \neq \phi\}$$

2.2. The class NP. It is natural to restrict attention to search problems in which a correct solution can be efficiently recognized, regardless of how it is reached.

Definition 1. An **NP** search problem is a search problem S for which there is an efficient algorithm that inputs an instance x and a purported solution for it y such that $|y| \leq |x|^{O(1)}$ and outputs 'yes' if and only if $(x,y) \in S$.

Definition 2 (The Class **NP**). The class **NP** is the set of languages $L \subseteq \{0,1\}^*$ that are decision versions of **NP** search problems.

A canonical example of a language in **NP** is Satisfiability (SAT). It consists of all satisfiable formulae φ. The corresponding search problem, described earlier as SATISFY, consists of all pairs (φ, a) where φ describes a logical Boolean formula, and a describes an assignment to the variables that satisfies the formula. Clearly, one can efficiently verify that a is a valid solution simply by plugging in the values and simplifying. Intuitively, this seems much easier to do than to actually find a from scratch. Indeed, an equivalent way to state the famous $\mathbf{P} \neq \mathbf{NP}$ conjecture is to say that there is no efficient way to always find a given φ.

We next describe an optimization problem called max-CSP, which is **NP**-complete. First, let us define a constraint.

Definition 3 (Constraint). Let $V = \{v_1, \ldots, v_n\}$ be a set of variables that take values in some finite alphabet Σ. A q-ary constraint $C = (\psi, i_1, \ldots, i_q)$ consists of a q-tuple of indices $i_1, \ldots, i_q \in [n]$ and a predicate $\psi : \Sigma^q \to \{0,1\}$. A constraint is *satisfied* by a given assignment $a : V \to \Sigma$ iff $\psi(a(v_{i_1}), a(v_{i_2}), \ldots, a(v_{i_q})) = 1$.

The CSP problem with parameters q and $|\Sigma|$ is defined as follows. The problem instance is a set V of variables, an alphabet Σ, and a set of constraints $C_1, \ldots, C_m$. The goal is to find an assignment to the variables that satisfies all of the constraints. Several well-known problems in **NP** are special cases of CSP. For example,

- 3-SAT is the problem when $\Sigma = \{0,1\}$, all constraints have $q = 3$ and a predicate that can be written as a disjunction of three literals, e.g. $\psi(a,b,c) = a \vee \neg b \vee c$.

- 3-COL is usually defined as a problem on graphs: given a graph, find a 3 coloring of the vertices $\chi : V \to \{1,2,3\}$ such that no two adjacent vertices are colored by the same color. Clearly, this is a special case of CSP if we take variable for each vertex, $q = 2$, $\Sigma = \{1,2,3\}$, and let each edge represent a constraint whose predicate is the unequal predicate $\psi(a,b) = 1$ iff $a \neq b$.

- 3-LIN is the CSP problem when $\Sigma = \{0,1\}$, all constraints have $q = 3$ and all predicates are affine equations over the field $GF(2)$.

The related optimization problem max-CSP is the problem of finding an assignment that maximizes the number of satisfied constraints.

2.3. NP and Efficient Proof Systems. An equivalent definition of the class **NP**, is as a class of efficient proof systems. Roughly speaking, a logic proof system consists of a set of axioms and inference rules, such that any of its theorems can be obtained through a sequence of steps each of which is either an axiom, or is inferred from previous steps through application of one of the inference rules. As an example, one should keep in mind Frege's propositional calculus (see [11]) which consists of six axioms and one inference rule. A proof system has two important properties called *completeness* and *soundness*. Completeness means that every provable statement is true, and soundness means that every true statement has a proof.

One can generalize this notion of a proof system as follows. First, observe that one can fully describe the proof system through its verification process, which we will call its *verifier* from now on. The verifier is nothing but an algorithm that checks that each step is either an axiom or a result of applying a derivation rule on some previous steps. Now, since the verifier fully defines the proof system, we generalize by allowing a wider class of verifiers. Indeed, we allow the verifier to be *any* efficient algorithm. We insist on the efficiency of the verifier to maintain the intuitive notion that checking a proof should be an easy and technical matter, unlike, perhaps, coming up with one.

More formally,

Definition 4. A language $L \subset \{0,1\}^*$, has an *efficient proof system* if there is an efficient algorithm, called a verifier, that inputs a string x and also a purported proof string y. The verifier and runs in time polynomial in $|x|$ and either accepts or rejects, and

- (Completeness:) If $x \in L$ then there is some y such that the verifier accepts. (In other words for every $x \in L$ there is an acceptable proof y).

- (Soundness:) If $x \notin L$ then for every y the verifier rejects. (In other words, no proof y will be able to prove a false statement).

In this proof system the set L is interpreted to be the set of all 'theorems', or 'true' statements.

Definition 5 (The class **NP**, second definition). The class **NP** is the set of all languages L that have efficient proof systems.

For example, the set SAT of all logical Boolean formulae that are satisfiable has an efficient proof system. The verifier is a simple algorithm that expects, as proof, a string y that represents a satisfying assignment, and then plugs it in the formula and simplifies. From this example it becomes clear that **NP** provides a rich variety of proof systems, quite different from the sequential ones with which we began our discussion.

Reductions and NP completeness One can move quite freely between different **NP** languages through *reductions*. A reduction from L_1 to L_2 is a mapping $r : \{0,1\}^* \to \{0,1\}^*$ that has two properties. First, it is computable in polynomial time, and second, it guarantees that $x \in L_1$ if and only if $r(x) \in L_2$. There are languages in **NP**, called **NP**-complete, that are "hardest" in the sense that any other **NP** language can be reduced to them. SAT is a canonical example for an **NP**-complete set, and there are many others. The **NP**-completeness of SAT implies that for every $L \in \mathbf{NP}$ there is a reduction r mapping it to SAT.

Through the notion of reductions we can see that one set can have many different NP verifiers (or proof systems). For example, let L denote the set of all graphs containing a clique of size at least k, for some k. Since $L \in \mathbf{NP}$ and since SAT is NP-complete, there is a reduction from L to SAT. This gives rise to the following (not so natural) verifier for L: Given an instance $G \stackrel{?}{\in} L$, the verifier will first run the reduction to compute the formula $r(G)$, and then check that the proof is a satisfying assignment for this formula.

This example demonstrates that one **NP** set can have many proof systems, quite different from one another. We will see in Section 3 that some of these proof systems turn out to have quite remarkable properties.

2.4. Error Correcting Codes.

The *Hamming* distance of a pair of strings $x, y \in \{0,1\}^n$ is denoted $\mathrm{Dist}(x,y)$ and is the number of bits on which they differ. The relative Hamming distance is denoted by $\mathrm{dist}(x,y)$ and is equal to $\mathrm{Dist}(x,y)/n$.

We define an error correcting code with relative distance δ to be a mapping $C : \{0,1\}^k \to \{0,1\}^n$ for which the following holds:

$$\forall x \neq y \in \{0,1\}^k, \qquad \mathrm{dist}(C(x), C(y)) \geq \delta.$$

We usually think of an error correcting code as part of an asymptotic family of codes C_k one for each message length k, but this will be suppressed from the following discussion.

Let us remark that error correcting codes with constant relative distance are not hard to come by. A $GF(2)$-linear mapping $C : \{0,1\}^k \to \{0,1\}^n$ with $n = \Theta(k)$ that is defined by an $n \times k$ matrix with 0/1 entries selected independently at random will have constant relative distance with overwhelming probability.

3. The PCP Theorem

Probabilistically Checkable Proofs (PCPs) have evolved from the celebrated notion of interactive proofs [23, 5] and the complexity class IP. This line of research was originally motivated by cryptography and the study of what it means for two entities to prove something to one another. Soon it lead to a list of remarkable complexity-theoretic results (e.g., see [30, 40, 9, 17, 7]), which seemed to suggest existence of a PCP verifier for every language in **NP**. At the same time, a surprising connection was discovered by [16], showing that existence of such a PCP verifier would imply that it is NP-hard to determine the size of the maximum clique in a graph, *even approximately*. With this additional motivation, the proof was soon found, first partially in [4] and then fully in [3], and came to be known as the PCP theorem (Theorem 1.1).

In this section we present the PCP theorem in two guises. First we follow the standard presentation, as done in the introduction, but in more formal details. Next, we present the PCP theorem as an NP-hardness result about approximating constraint satisfaction problems.

3.1. The PCP theorem - formal statement. The PCP theorem [4, 3] describes, for every language $L \in \mathbf{NP}$, a proof system in which the verifier is both enhanced with additional randomness and restricted in its access to the proof.

Notation For any $n \in \mathbb{N}$ we denote by $[n]$ the set of n elements $\{1, \ldots, n\}$. For a string $s \in \{0,1\}^n$ and a set of indices $I = \{i_1 < \cdots < i_t\} \subset [n]$, we denote by $s_{|I}$ the t-bit string $s_{i_1} s_{i_2} \cdots s_{i_t}$ obtained by restricting s to I.

We now define formally the class $PCP[r, q]$ through the notion of an (r, q)-verifier.

Definition 6. An (r, q)-verifier for a language $L \in \mathbf{NP}$ is given an input $x \in \{0,1\}^n$ and is also allowed to read r random bits. It then computes a set of q indices $I = \{i_1, \ldots, i_q\}$ and a Boolean predicate $\psi : \{0,1\}^q \to \{0,1\}$ and accepts if and only if $\psi(\pi_{|I}) = 1$.

Definition 7. The class $PCP_{c,s}[r, q]$ contains all languages L for which there is an $(O(r), O(q))$-verifier V such that

- (Completeness:) If $x \in L$ then there is a proof π such that
$$\Pr[\psi(\pi_{|I}) = 1] \geq c,$$
- (Soundness:) If $x \notin L$ then for any proof π,
$$\Pr[\psi(\pi_{|I}) = 1] \leq s.$$

where the probability is over the r bits of randomness of the verifier that are used to compute ψ, I.

The PCP theorem says that every language in NP has a verifier that uses at most $O(\log n)$ random bits and reads only $O(1)$ bits from the proof. Here n denotes the input length and the $O(\cdot)$ notation refers to asymptotic growth of $n \to \infty$. In other words,

Theorem 3.1 (PCP Theorem, [4, 3]). $\text{NP} \subseteq \text{PCP}_{1,\frac{1}{2}}[O(\log n), O(1)]$.

3.2. The PCP theorem – a hardness of approximation result.

The beautiful connection discovered by Feige et. al. [16] shed new light on the hardness of approximating combinatorial optimization problems. (A formal definition of approximation and optimization problems can be found in Section 2). This entire field was soon completely transformed, when many known algorithms found nearly matching lower bounds via the PCP theorem.

Approximation algorithms are a natural way to cope with **NP**-hard problems. By the late 1980's approximation algorithms have been developed for a variety of NP-hard problems. Different problems were found to have approximation algorithms with vastly differing values of the approximation ratio r. Predating the discovery of the PCP theorem and the connection of [16], there were no lower bounds on approximation: it seemed possible that approximation is never NP-hard when $r > 1$, and even that every **NP**-problem can be approximated up to any precision, in polynomial time.

An extreme example is the difference between the approximation behavior of minimum vertex cover[2] and maximum independent set[3]. In any graph $G = (V, E)$, if S is an independent set then $V \setminus S$ is a vertex cover. Thus finding an exact solution for one problem is the same as for the other. In contrast, the best approximation for the maximum independent set is within a factor only slightly below the trivial factor of n, whereas vertex cover can be 2-approximated quite easily.

The PCP theorem implied, for the first time, that numerous problems (including, for example, the problems mentioned above) are hard to approximate to within some constant factor. This has had a tremendous impact on the study of combinatorial optimization problems, and today the PCP theorem stands at the heart of nearly all hardness-of-approximation results.

The equivalence between the PCP theorem (as stated in Theorem 3.1) and a hardness of approximation result is easily described in terms of the constraint satisfaction problem CSP (see definition in Section 2). First, let us state a typical hardness of approximation result. We will then prove that it is equivalent to the PCP theorem.

Theorem 3.2. *For every $L \in$ **NP** there is an absolute constant $q \in \mathbb{N}$ and a reduction that maps $x \overset{?}{\in} L$ to a* CSP *instance with $|\Sigma| = 2$ and q-ary constraints such that if $x \in L$ then there is an assignment satisfying all constraints, and if $x \notin L$ then every assignment satisfies at most $\frac{1}{2}$ of them.*

[2] In the minimum vertex cover problem, one is given a graph and needs to find a smallest set of vertices that touch all edges.

[3] In the maximum independent set problem, one is given a graph and needs to find a largest set of vertices that spans no edges.

Proposition 3.3 ([16]). *Theorem 3.1 and Theorem 3.2 are equivalent.*

Proof. ($\Rightarrow$): Let $L \in \mathbf{NP}$ and let Ver be the (r, q)-verifier for L with $r = O(\log |x|)$ and $q = O(1)$. Recall that for each random string ρ, when Ver is run on the input x with randomness ρ, it computes a predicate and a set of indices, i.e., a q-ary constraint. The reduction will simulate Ver on every possible random string, thus generating a list of $2^{O(\log n)}$ constraints over variables that represent the proof bits. This is the output CSP instance of the reduction. It is easy to see that the completeness and soundness of Ver translate to the desired behavior of the CSP instance.

($\Leftarrow$): Let $L \in \mathbf{NP}$, we design an $(O(\log n), O(1))$-verifier for it. The verifier will input x, run the reduction computing from x a CSP instance, and then expect the proof π to contain an assignment to the variables of the CSP instance $\mathcal{C} = \{C_1, \ldots, C_m\}$. The verifier will use its randomness select a random constraint $C_i = (\psi, i_1, \ldots, i_q) \in \mathcal{C}$, read the corresponding bits from the proof and accept iff $\psi(\pi_{|\{i_1,\ldots,i_q\}}) = 1$. □

We discuss the proof of this theorem in Section 5.

3.3. Further Results. One way to study the class **NP** is by examining its limits, or "boundary". The search for tightest parameters and parameter tradeoffs of PCP verifiers that still capture **NP** has been the focus of research in the past two decades. Some questions of particular interest are

- Suppose the verifier is restricted to make exactly $q \geq 2$ queries. What is the smallest possible probability of error ? (this refers to the probability that the verifier accepts an $x \notin L$, or rejects an $x \in L$).
- Viewing a PCP as an encoding of an NP proof, what is the smallest possible encoding length? For example, could there be a mapping that takes an n bit NP proof into an $O(n)$ bit PCP ? Currently, the shortest known PCPs take n bit **NP** proofs to PCPs of length $n \cdot (\log n)^{O(1)}$.

In terms of inapproximability, similar efforts have been made. Here, one is interested in finding, for each approximation problem, what is the largest r for which it is still **NP**-hard[4] to r-approximate the problem. In several cases this r matches the best known approximation algorithm, and in other cases there is still a huge gap.

The Unique Games Conjecture. One direction that has been very successful in recent years stems from the unique games conjecture of Khot [25]. This conjecture says that a certain restricted type of CSP instance is **NP**-hard. This can be viewed as a conjectured "strengthening" of the PCP theorem. There have been many works [28, 27, 14, 29, 12, 39, 35] showing that if this conjecture were true, then various approximation problems would be even harder to approximate, often to within factors that match their best approximation algorithms. Very recently

[4] A problem is NP-hard if an algorithm for can be used to solve any other problem in **NP**.

Arora et. al [2] found a slightly subexponential algorithms for unique games. This can be taken as evidence that the unique games CSP may not be **NP** hard, and in the least, it seems easier than other CSP's such as 3-SAT.

4. Local Testing and Stability

The idea that global phenomena can be determined based on local behavior is commonplace. Whether it is astronomers that study the universe through observing a tiny fraction of it, or statisticians deducing about entire populations from polled data. Even when local observations are somewhat noisy we still manage to deduce global properties quite nicely, or at least so we think. In fact, what makes this paradigm go through is the fact that these properties are *stable*.

Generally speaking, a system of constraints (e.g. equations) characterizes a set in a stable manner if any approximate solution to the system is nothing but a perturbation of some exact solution.

A system of constraints over Boolean variables is *stable* if any solution that satisfies many of the constraints must be close to a solution that satisfies the entire system.

In theoretical computer science the focus shifts to the solution set itself rather than the constraints that characterize it. A set that can be characterized by a stable set of local constraints (i.e., where each constraint looks at only at most q bits of the proposed solution) is called *locally testable*.

In what follows we will formally define local testability. Next, we will give an example of a locally testable property. We will then define and discuss locally testable codes, which are an important notion in the construction of PCPs. Finally, we will discuss connections to other mathematical areas in which similar stability phenomena are studied.

4.1. Local Testing. A property of binary strings is a subset $L \subset \{0,1\}^*$, alternatively described as a sequence $(L_n)_{n\in\mathbb{N}}$, where $L_n \subseteq \{0,1\}^n$. We next define what a locally testable property is. First let us recall that a constraint over a string $x \in \{0,1\}^n$ is defined by a predicate $\psi : \{0,1\}^q \to \{0,1\}$ and q indices $i_1, \ldots, i_q \in [n]$ and is satisfied on string $x \in \{0,1\}^n$ if $\psi(x_{i_1}, \ldots, x_{i_q}) = 1$.

Definition 8 (Local testability). A property $L_n \subset \{0,1\}^n$ is locally testable with q queries and error ϵ if there is a set of q-ary constraints $C_1, \ldots C_m$ over n bits such that for every $x \in \{0,1\}^n$

- If $x \in L$ then $\Pr_{j\in[m]}[x \text{ satisfies } C_j] = 1$
- If $x \notin L$ then $\Pr_{j\in[m]}[x \text{ satisfies } C_j] \leq \epsilon$

A property $(L_n)_{n\in\mathbb{N}}$ is locally testable if L_n is locally testable for every $n \in \mathbb{N}$.

What is it that makes a property testable? Questions regarding what types of properties are locally testable are the topic of a field called *property testing*. This

field started out from works on testing low algebraic degree of functions [4, 3, 16, 37, 19, 6, 33] similar to the example below, and more recently has evolved into property testing of graph properties [21], codes [37, 22], and various other types of objects. For a survey, see [20, 36].

4.2. Example: low degree testing. In this section we describe a property that is locally testable. The purpose is to give a sense of what it means to be locally testable. As our example, we chose the linearity testing of [10] which also plays a role in the actual construction of PCPs. We will return to this example later when we describe the PCP construction.

Let our universe consist of all functions $f : \mathbb{F}_2^k \to \mathbb{F}_2$, where $\mathbb{F}_2 = \{0, 1\}$ is the field with two elements, and $k \in \mathbb{N}$. The property under consideration is the subset of all *linear* functions:

$$\text{LIN} = \left\{ f : \mathbb{F}_2^k \to \mathbb{F}_2 \;\middle|\; \exists a_1, \ldots, a_k \in \mathbb{F}_2,\ s.t.\ \forall x\ f(x) = \sum_{i=1}^{k} a_i x_i \right\}.$$

Although we only defined local testability for properties of strings, Definition 8 easily extends to functions by identifying a function $f : \mathbb{F}_2^k \to \mathbb{F}_2$ with the string $f \in \{0, 1\}^{2^k}$ that describes its truth table.

In order to show that the property LIN is testable, we must find a set of local constraints that characterize the property in a stable way. This seems easy: for each pair of points $x, y \in \mathbb{F}_2^k$ think of the constraint that is satisfied if and only if

$$f(x) + f(y) = f(x + y).$$

Denoting this constraint by $C_{x,y}$ we define $\mathcal{C} = \{C_{x,y}\}_{x,y \in \mathbb{F}_2^k}$ to be our system of constraints. It remains to verify that the definition of testability holds.

For sanity check we observe that if f is linear then every constraint will be satisfied. Moreover, if every constraint is satisfied then surely the function is linear. What is less obvious is what happens when f satisfies almost all of the constraints, but not quite all. Does it necessarily have to be close to some linear function?. A priori, one could imagine a function g for which

$$\Pr_{x,y \in \mathbb{F}_2^k}[g(x) + g(y) = g(x + y)] > 0.99 \tag{1}$$

and yet g is far from every linear function.

Nevertheless, the linearity testing theorem of [10] implies that any g for which (1) holds must agree with some linear function on at least 99% of the domain.

Theorem 4.1 ([10, 8]). *Let $g : \mathbb{F}_2^k \to \mathbb{F}_2$ be a function for which $\text{dist}(g, \text{LIN}) \leq \frac{1}{2}$. Then*

$$\Pr_{x,y \in \mathbb{F}_2^k}[g(x) + g(y) \neq g(x + y)] \geq \text{dist}(g, \text{LIN}).$$

The proof of this theorem turns out to be a relatively simple exercise in discrete Fourier analysis. However, even straightforward generalizations of it (for example, for the property of functions having degree $2, 3$, etc.) require significantly more work [1, 38, 24, 42] and many open questions still remain.

4.3. Locally testable codes. Let us return to the main topic of this paper, Probabilistically Checkable Proofs. Recall our attempt to construct probabilistically checkable proofs by encoding the **NP** proof. This encoding should amplify errors in the original proof had there been any. The PCP verifier must check that the given proof string is valid, i.e., that it is a valid encoding of a valid **NP** proof. Focusing on the first part of the requirement (i.e., that of being a valid encoding) is the task of locally testing a code.

Definition 9 (Locally testable code). A locally testable code is an error correcting code $C : \{0,1\}^k \to \{0,1\}^n$ whose image $Im(C) = \left\{ C(x) \;\middle|\; x \in \{0,1\}^k \right\}$ is locally testable.

One usually considers error correcting codes with large relative distance, $\delta = \Omega(1)$. In such cases, every bit in the encoding should depend, on average, on a constant fraction of the message bits. In contrast, the fact that $Im(C)$ is locally testable means that there are very local correlations between the encoding bits. These two requirements are in tension with one another, and this is partly what makes the construction of locally testable codes more challenging.

There are few known constructions of LTCs with reasonable parameters. A first example is the Hadamard code

$$H : \{0,1\}^k \to \{0,1\}^{2^k}$$

that encodes a message $a = (a_1, \ldots, a_k) \in \{0,1\}^k$ by a string $H(a) \in \{0,1\}^{2^k}$ that is the truth table of the linear function f defined by $f(x) = \sum_{i=1}^{k} a_i x_i$. First, note that if $a \neq b \in \{0,1\}^k$ then $\mathrm{dist}(H(a), H(b)) = \frac{1}{2}$, so this code has good relative distance. Next, note that H is locally testable. This follows from the testability of the property LIN described in the previous section.

The main drawback of the Hadamard code as an LTC is its encoding length, encoding k bits by 2^k. There are much more efficient constructions, yet they are much more complicated and less 'explicit'. In general these come by stripping down a construction of an equivalent PCP. It is a challenging question to find a construction of an LTC that is as explicit as the Hadamard code, yet with better parameters.

4.4. Connections with other fields. Questions about stability of systems appear in various other fields of mathematics. Below we describe a few examples that have some direct connections with PCPs.

4.4.1. Approximate Polynomials. Polynomial functions obey local constraints that come essentially from interpolation formulae. For example, a degree d univariate polynomial obeys many constraints on $d+2$ points: simply use the first $d+1$ points to compute the value on the remaining point, and test that this is indeed the value. That these constraints are also stable is the topic of "low degree tests" which play a key role in the proof of [4, 3] of the PCP theorem.

Similarly a multi-variate polynomial of degree d that must behave in a certain way on subspaces of dimension $d+1$ (again due to interpolation). The fraction

of subspaces on which a function behaves like a polynomial is exactly captured by the so-called Gowers $d+1$ uniformity norm.

A sequence of works [1, 38, 24, 42] has been focused on characterizing what functions have Gowers uniformity norm that is strictly above the value the norm of a random function. This is called the inverse conjecture for the Gowers uniformity norm, see also [41]. These questions are related to questions in arithmetic combinatorics which study the behavior of sets containing many arithmetic progressions. In PCPs, such results have been used for constructing PCPs with near optimal tradeoff between the number of queries and the error probability [39].

4.4.2. Approximate Dictatorships. Dictatorships are functions $f : \{0,1\}^n \to \{0,1\}$ that depend on only one variable. There are n such basic functions, $\chi_i(x_1, \ldots, x_n) = x_i$ for each $i \in [n]$. There are many different ways to characterize dictatorships in the hypercube, and each way leads to a different, and often interesting, stability question. Here are two examples.

- One can measure the *average sensitivity* of a function. This is the probability that $f(x) = f(x+e_i)$ when $x \in \{0,1\}^n$ and $i \in [n]$ are chosen at random (and e_i is the unit vector with 1 in the i-th coordinate). For balanced functions this value is minimized on dictators. A stability question is to characterize all functions with average sensitivity that is within a constant factor of the minimum. Friedgut [18] proved that such functions must be close to "juntas" which are functions that depend on a constant number of their n variables. Friedgut's theorem has been used in results related to the hardness of approximating the minimum vertex cover in a graph [15, 28].

- The "majority is stablest" theorem is concerned with the noise sensitivity of Boolean functions. This is the probability that $f(x) = f(y)$ when y is a "noisy" copy of x, i.e. when each coordinate of x is flipped with probability ϵ. Dictatorships are the least sensitive to noise, and the conjecture above says that every function that is "far" from being a dictatorship must have sensitivity at least as much as the majority function does (the majority function evaluates to 1 on inputs x that have more 1's than 0's). The "majority is stablest" theorem was conjectured in [27] as part of an inapproximability result about MAX-CUT. It was later proved in [32] and lead to the discovery of a powerful 'invariance principle' [32, 31]. At the heart of these results one needs a method to differentiate between dictatorships and between functions that are "smooth" and have no variable that has large influence. The 'invariance principle' is a generalization of the central-limit-theorem showing that smooth polynomials behave almost the same regardless of how each individual variable is distributed.

  Raghavendra [35] relied on the invariance principle to prove a very general inapproximability result for CSP's.

We point the reader to Khot's article [26] for more illuminating examples.

5. Construction of Probabilistically Checkable Proofs

The original proof and formulation of the PCP theorem came from study of proof verification. The techniques are largely based on algebraic encodings and testing results that are generally called "low degree tests". More recently, a combinatorial proof was given by the author [13]. This proof is framed more naturally as a hardness of approximation result, and it relies on rapid mixing of random walks on expanding graphs. In this section we sketch these two approaches.

5.1. PCPs using algebra. The original proof of the PCP theorem, relies on "algebraic" encodings of **NP** witnesses by low degree functions. This proof proceeds by constructing a $(O(\log n), O(1))$-verifier for every language in **NP**, thus proving Theorem 3.1.

5.1.1. A verifier for linear CSPs. Before constructing a $(\log n, 1)$-verifier for every **NP** language, let us construct such a verifier for the CSP language defined by linear and affine predicates. In other words, the input is a set of linear or affine constraints, say each over two variables, and the goal is to verify that a given assignment a satisfies all constraints. This is really only a baby case since a verifier can determine efficiently whether the system is satisfiable without looking at a proof at all. Nevertheless, it gives some intuition for the actual proof.

We can encode the assignment a that satisfies all of these constraints using the so-called *Hadamard* code, that was described in Section 4.3. The PCP verifier would expect as proof the encoding $H(a)$ of an assignment. Since the Hadamard code is a locally testable code, the verifier can test that a purported proof w is a valid encoding of some a. Moreover, it is not hard to see, that since this encoding consists of all linear forms in the input message a, it is also easy to test whether a satisfies some set of affine predicates.

5.1.2. The general case. Moving on to the general case, here are some of the points that are resolved along the way.

- First, we need to be able to encode non-linear predicates. It turns out that it suffices to consider degree 2 predicates since these are already expressive enough to capture **NP** (in other words, the CSP problem with degree 2 predicates is **NP**-complete).
- Our second concern is the exponential length of the Hadamard encoding. The PCP encoding should be efficient. In particular, we cannot afford to encode n bits of message by 2^n bits, as done by the Hadamard code. This again is resolved by considering polynomials of higher degree, say $d = \log n$, over a larger field (of size say $(\log n)^{O(1)}$). In other words, the encoding of an n-bit assignment would be the point evaluation of a polynomial function $p : \mathbb{F}^m \to \mathbb{F}$ whose restriction to some predefined set of points $S \subset \mathbb{F}^m$ agrees with the original assignment. This brings the length of the encoding down, but causes a new problem. Although "low degree tests" for such functions

have been proven, these tests necessarily use at least d queries to test whether a function has degree d. This is no longer constant when $d = \log n$.

- In order to reduce the number of queries from $\log n$ to $O(1)$ one relies on several steps and most importantly, on *composition*. We do not describe this here.
- Finally, we neglected to describe how to check that the encoded proof encodes a *valid* proof, i.e., one that satisfies the original constraints. This is done by additional machinery and in particular using a so-called sum-check procedure. For details see [4, 3].

5.2. PCPs using random walks on graphs. We now describe a combinatorial proof of the PCP theorem due to the author, see [13, 34]. This proof is best described as an inapproximability result, i.e., as a proof for Theorem 3.2, which we quote again for convenience:

Theorem 3.2. *For every $L \in \mathbf{NP}$ there is a $q \in \mathbb{N}$ and a reduction that maps $x \overset{?}{\in} L$ to a* CSP *instance with $|\Sigma| = 2$ and q-ary constraints such that if $x \in L$ then there is an assignment satisfying all constraints, and if $x \notin L$ then every assignment satisfies at most $\frac{1}{2}$ of them.*

It is enough to fix L to be one **NP**-complete language, say $3 - \text{COL}$. Recall that in this problem the input is a graph $G = (V, E)$ and the goal is to decide whether there is a coloring $c : V \to \{1, 2, 3\}$ such that for every $(u, v) \in E$, $c(u) \neq c(v)$.

We construct an algorithm that inputs a graph G and outputs a new graph G' such that

- If G is 3 colorable then so is G'.
- If G is not 3 colorable, then every coloring of the vertices of G' must have at least some $\epsilon > 0$ fraction of unsatisfied (i.e., monochromatic) edges.

Let the *unsat* value of a graph G, denoted $\text{unsat}(G)$, be the minimum fraction of monochromatic edges, when going over all possible 3-colorings of G

$$\text{unsat}(G) = \min_{c:V\to[3]} \left[\Pr_{(u,v)\in E} [c(u) = c(v)] \right].$$

Note that G is 3-colorable if and only if $\text{unsat}(G) = 0$. If G is not 3-colorable then surely $\text{unsat}(G) \geq 1/|E|$.

Our algorithm proceeds by a sequence of encodings,

$$G \to G_1 \to G_2 \to \cdots \to G'$$

where the unsat value is amplified a little in each step.

The basic transformation $G_i \to G_{i+1}$ will amplify this value by a constant multiplicative factor. This will be done without harming the 3-colorability of G in case it was 3-colorable. In other words, if G_i is 3-colorable, then so is G_{i+1}, but

otherwise $\text{unsat}(G_{i+1}) \geq 2 \cdot \text{unsat}(G_i)$ (unless $\text{unsat}(G_i)$ exceeds some constant threshold).

After repeating this basic step $O(\log n)$ times the unsat value will become some absolute constant and we are done. The transformation taking G_i to G_{i+1} only causes a linear increase in the size of G, so repeating it this many times ($O(\log n)$) will not cause the output to be too large.

5.2.1. Amplifying the *unsat* value. Let us sketch a description of the transformation taking G_i to G_{i+1}. For notation convenience we denote the input and output graphs of the transformation by G, H instead of G_i, G_{i+1}.

This transformation involves two main steps.

1. In the first step G is encoded by a graph G' and a 3-coloring for G is encoded by a k-coloring for G', where $k > 3$ is some constant. The vertices of G' are the same as those of G, and the color of a vertex in G' is supposed to encode the colors of all of its neighbors in G. The constraints on the edges of G' are not "inequality" constraints as in a proper k-colorability problem, but rather more general constraints that check that the local colorings are consistent with each other. E.g., if two vertices assign a different color to a common neighbor then this is an inconsistency.

   Finally, we place an edge between two vertices in G' if they are at distance up to 100 from each other.

   By construction, if G were 3-colorable, then there is a k-coloring that satisfies all of the new constraints. The main thing to prove is the converse. Under some (expansion) conditions on the structure of G, one can show that if the unsat value of G was α, the fraction of unsatisfiable constraints on G' is at least 2α.

2. The second step involves an alphabet reduction, taking the k-colorability instance G' back into a 3-colorability instance H without harming the *unsat* value too much. This step relies on composition similarly to the way it is applied in the original proof of the PCP theorem, and is beyond our scope.

Acknowledgements

I wish to thank Adi Akavia, Elazar Goldenberg, Or Meir, Igor Shinkar and Oded Schwartz for helpful comments.

References

[1] N. Alon, T. Kaufman, M. Krivelevich, S. Litsyn, and D. Ron. Testing Reed-Muller codes. *IEEE Transactions on Information Theory*, 51(11):4032–4039, 2005.

[2] S. Arora, B. Barak, and D. Steurer. Subexponential algorithms for unique games and related problems. submitted, 2010.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[4] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[5] L. Babai. Trading group theory for randomness. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 421–429, 1985.

[6] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 21–31, 1991.

[7] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[8] M. Bellare, D. Coppersmith, J. H. stad, M. Kiwi, and M. Sudan. Linearity testing over characteristic two. *IEEE Transactions on Information Theory*, 42(6):1781–1795, 1996.

[9] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi prover interactive proofs: How to remove intractability assumptions. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 113–131, 1988.

[10] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 73–83, 1990.

[11] S. Buss. An introduction to proof theory. *Handbook of proof theory*, pages 1–78, 1998.

[12] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Computational Complexity*, 15(2):94–114, 2006.

[13] I. Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3), 2007.

[14] I. Dinur, E. Mossel, and O. Regev. Conditional hardness for approximate coloring. *SIAM Journal on Computing*, 39(3):843–873, 2009. Prelim version in STOC 2006.

[15] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005. Preliminary version appeared in STOC 2002.

[16] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. *Journal of the ACM*, 43(2):268–292, 1996.

[17] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 1994.

[18] E. Friedgut. Boolean functions with low average sensitivity depend on few coordinates. *Combinatorica*, 18(1):27–35, 1998.

[19] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 32–42, 1991.

[20] O. Goldreich. Introduction to testing graph properties. Manuscript, 2010.

[21] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation abstract. *Journal of the ACM*, 45(4):653–750, 1998.

[22] O. Goldreich and M. Sudan. Locally testable codes and PCPs of almost-linear length. *J. of the ACM*, 53(4):558–655, 2006.

[23] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proofs. *SIAM Journal on Computing*, 18:186–208, 1989.

[24] B. Green and T. Tao. An inverse theorem for the gowers U3 norm. arXiv:math/0503014v3, 2005.

[25] S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775. ACM Press, 2002.

[26] S. Khot. Inapproximability of NP-complete problems, discrete fourier analysis, and geometry. 2010.

[27] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM J. Computing*, 37:319–357, 2007.

[28] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. In *Proc. of 18th IEEE Annual Conference on Computational Complexity (CCC)*, pages 379–386, 2003.

[29] S. Khot and N. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ_1. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 53–62, 2005.

[30] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, October 1992.

[31] E. Mossel. Gaussian bounds for noise correlation of functions. *GAFA*, 19:1713–1756, 2010.

[32] E. Mossel, R. O'Donnell, and K. Oleszkiewicz. Noise stability of functions with low influences: invariance and optimality. *Ann. Math.*, 171(1):295–341, 2010.

[33] A. Polishchuk and D. Spielman. Nearly linear size holographic proofs. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 194–203, 1994.

[34] J. Radhakrishnan and M. Sudan. On Dinur's proof of the PCP theorem. *Bulletin of the AMS*, 44:19–61, 2007.

[35] P. Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *Proc. 40th ACM Symp. on Theory of Computing*, pages 245–254, 2008.

[36] D. Ron. Property testing (a tutorial). *Handbook on Randomization*, 2, 2001.

[37] R. Rubinfeld and M. Sudan. Testing polynomial functions efficiently and over rational domains. In *Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 23–32, 1992.

[38] A. Samorodnitsky. Low degree tests at large distances. In *Proc. 39th ACM Symp. on Theory of Computing*, 2007.

[39] A. Samorodnitsky and L. Trevisan. Gowers uniformity, influence of variables, and PCPs. In *Proc. 38th ACM Symp. on Theory of Computing*, pages 11–20, 2006.

[40] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992. Prelim. version in 1990 FOCS, pages 11–15.

[41] T. Tao. The inverse conjecture for the Gowers norm over finite fields via the correspondence principle. Blog post on http://terrytao.wordpress.com, 2008.

[42] T. Tao and T. Ziegler. The inverse conjecture for the gowers norm over finite fields via the correspondence principle. *Analysis and PDE*. to appear, see arXiv:0810.5527.

The Weizmann Institute of Science Rehovot, 76100 ISRAEL
E-mail: irit.dinur@weizmann.ac.il