BERNARD CHAZELLE

The PCP theorem

Astérisque, tome 290 (2003), Séminaire Bourbaki, exp. nº 895, p. 19-36
<http://www.numdam.org/item?id=SB_2001-2002__44__19_0>

Séminaire BOURBAKI
54^e année, 2001-2002, n^o 895, p. 19 à 36

Novembre 2001

THE PCP THEOREM
[after Arora, Lund, Motwani, Safra, Sudan, Szegedy]

by **Bernard CHAZELLE**

1. INTRODUCTION

The notion of *interactive proof systems* evolved out of cryptography and computational group theory. The cryptographic context is best explained through a little tale (perhaps one day to come true). One fine morning, one of your esteemed colleagues wakes up with, in his head, a crisp, concise, complete proof of Riemann's Hypothesis! Wisdom being one of his many qualities, he is not about to post his proof on the internet. Paranoia being another one, he is not even willing to reveal a single bit of information about the proof; that is, besides its conclusion that the RH is true. Is there any way for your colleague to convince you and the rest of the mathematical community that, indeed, he has a correct proof? Of course, one needs to define what exactly is meant by not "revealing a single bit". That is the subject of *zero-knowledge* cryptography.

The *PCP theorem* addresses a simpler variant: Can your colleague write down his proof in such a way that, were you to peek into it at a constant number of randomly chosen spots, you would leave utterly convinced of its validity? In other words, can he encode the proof as a string of bits so that: (i) a correct proof will never fail to convince you; (ii) an incorrect one will fool you with only a negligible probability? The catch is, you will be allowed to look at only a constant number of bits chosen at random. The PCP theorem asserts the existence of such an encoding. It is striking that the number of lookups can be kept constant regardless of the length of the proof. In fact, if you can put up with a failure rate slightly above 1/2, i.e., accept a wrong proof half the time, but still never reject a correct one, then the number of bits can be reduced to 3. On the other hand, if you are allowed to read as many bits as are needed to store, say, two lines of this article, the probability of failure drops to 10^{-100}. A key point is that the new proof can be derived from the old one purely syntactically. In other words, one can write a compiler to translate the proof mechanically without any knowledge of mathematics. Furthermore, the new proof is not much longer than the previous one.

A common initial reaction to the PCP theorem is that it must be either wrong or trivial. Why wrong? It seems to imply that any flaw in the proof should spread itself all over the place, so as to be caught immediately in a random peek. But, how can so much information be stored in so few bits? Here is how: If the proof is correct, print it as such; if it is wrong, then intersperse the statement $2+2=3$ at every other step. The problem with that encoding is that a correct proof will not convince anyone. The beauty of the PCP theorem is not that flaws are caught so easily: it is that the mere absence of a flaw is persuasive in and of itself. There is nothing amazing about catching a liar's lie. But it is quite a feat to hear a true story from a congenital liar and end up believing it.

2. THE PCP VIEW OF NP

A *Turing machine* is a computer model whose main feature, for our purposes, is to be universal: in particular, whatever it can compute in time polynomial in the length of the input is believed to constitute what is tractable in *any* (non-quantum) model. The class P consists of the sets for which membership can be decided by a Turing machine in polynomial time. For example, the set of singular square integer matrices is in P, because determinants can be computed in a polynomial number of steps. The class NP includes the sets for which membership can be verified in polynomial time. For instance, the set of polynomials in $\mathbf{Q}[X_1, \ldots, X_n]$ with at least one zero in $\{0,1\}^n$ is in NP. The reason is that, given a polynomial f and a point $x \in \{0,1\}^n$, one can check if $f(x) = 0$ in time polynomial in the number of bits needed to represent f. To find such a zero from scratch seems more difficult (to put it mildly), and it is widely conjectured that P $\neq$ NP. Within computer science, this open question dwarfs all others in importance.

A 3-CNF formula is a conjunction of clauses, each one consisting of three literals; for example, $(v_1 \vee \neg v_2 \vee v_3) \wedge (v_2 \vee v_3 \vee \neg v_4)$. It is satisfiable if some true/false assignment of the v_i's makes the formula true. The one above is, whereas $(v_1 \vee v_1 \vee v_1) \wedge (\neg v_1 \vee \neg v_1 \vee \neg v_1)$ is not. The set of satisfiable 3-CNF formulas is called 3-SAT. A classical result of Cook and Levin says that 3-SAT is *NP-complete*, meaning that, not only it is in NP, but deciding membership in *any* NP set can be reduced to testing the satisfiability of a 3-CNF. The Cook-Levin theorem shows that to understand 3-SAT is to understand all of NP.

Many other sets are known to be NP-complete: for example, the set of 3-colorable graphs. (A graph is 3-colorable if its nodes can be colored red, white, and blue with no edge sharing the same color.) The existence of NP-complete sets brings breathtaking universality into the computing picture. It implies that anyone who can quickly color graphs can also solve algebraic equations over finite fields, factor integers, compute discrete logarithms, find short vectors in lattices, determine the largest clique in a graph, etc.

To formalize what a mathematical proof has to do with NP takes some effort, but the intuition is clear. In any reasonable axiomatic system, this set is in NP:

$$\{\langle T.1^n\rangle \mid T \text{ is a theorem with a proof of size at most } n\},$$

where $\langle T.1^n\rangle$ denotes the 0/1 string formed by writing the theorem T in binary in the axiomatic system and appending n ones at the end. A prover can guess a proof of length at most n, and the verifier can then check it in time polynomial in its length.

The class NP can be described in the language of proofs. If $L \in$ NP then, given any $x \in L$, there exists a *short proof*, i.e., a polynomial-time computation, that x indeed belongs to L; for example, the solution of an algebraic equation. Conversely, if $x \notin L$, then no proof can convince anyone that x is in L. *Probabilistically checkable proofs* (PCP) add a small twist to this view: randomization. A PCP system for a set L consists of a string of bits (the proof) and a Turing machine with access to random bits (the verifier). Given an input x of n bits, the verifier generates $r(n)$ random bits(1); then it looks up $q(n)$ bits of the proof at locations of its choice. The lookups are done all at once nonadaptively. Finally, after a polynomial amount of (deterministic) computation, the verifier must either accept or reject the proof. The class of sets L that satisfy the two requirements below is denoted by $\mathrm{PCP}[r(n), q(n)]$:

– Given any $x \in L$, there is a proof that causes the verifier to accept x with probability 1.
– Given any $x \notin L$, every proof is rejected with probability at least 1/2.

The functions r and q are called the *random-bit complexity* and *query-bit complexity*, respectively. To alleviate the notation, both of them are understood up to a constant factor. If $r(n) = O(\log n)$, the number of distinct random strings is polynomial and, by running the verifier on all of them, it is immediate that $\mathrm{PCP}[\log n, 1] \subseteq \mathrm{NP}$. Proving the reverse inclusion requires a great deal of ingenuity. The purpose of this article is to explain the proof at a conceptual level, leaving mathematical technicalities aside. The *PCP theorem* states that

$$\mathrm{NP} = \mathrm{PCP}[\log n, 1]. \tag{1}$$

Note that the proof size can be assumed to be polynomial since at most $q(n)2^{r(n)} = n^{O(1)}$ bits of the proof have a chance of ever being read. The PCP theorem can be restated in a way that highlights its "error-spreading" aspect. Given any 3-CNF formula Φ on n variables, there exists another one, denoted by Ψ, which contains $n^{O(1)}$ variables and is satisfiable if and only if Φ is. Furthermore, if Ψ is not satisfiable, then no truth assignment can satisfy more than a fraction $1 - \varepsilon$ of its clauses, for some constant $\varepsilon > 0$. Finally, Ψ can be derived from Φ in polynomial time.

(1)Throughout our discussion, random points or numbers are drawn uniformly, independently from a set that is always clearly understood from the context; in this case the set is $\{0, 1\}$.

It is instructive to see how this follows from (1), because the argument anticipates aspects of the proof of the PCP theorem. Consider a PCP system for Φ. Among the $2^{r(n)}$ possible random strings, some lead to acceptance, others (possibly) to rejection. Given such a string s, let $\Pi_1, \ldots, \Pi_q$ be the bits of the proof read by the verifier. (The locations of these bits depend on s but not on the proof itself.) Let Φ_s be a Boolean formula that evaluates to true if and only if $\Pi_1, \ldots, \Pi_q$ lead to acceptance: Φ_s has $q = O(1)$ variables, each one corresponding to one of the bits read. It is routine to convert Φ_s into a constant size 3-CNF formula Φ'_s by adding a few auxiliary variables if necessary. The formula $\Psi = \bigwedge_s \Phi'_s$ fits the bill. To see why, consider the (only interesting) case: If Φ is not satisfiable, then regardless of the proof, i.e., of the truth assignment of the Φ_s's variables, at least half of these formulas are false and, hence, so is a constant fraction of the clauses in Ψ. □

This characterization of the PCP theorem, which interestingly makes no mention of proofs, verifiers, or even randomization, points to the connection between PCP and inapproximability. Indeed, it implies that it is NP-complete to distinguish between a satisfiable formula and one for which no truth assignment satisfies at least a fraction $1 - \varepsilon$ of the clauses. Another way to look at this result is that if we set out to maximize the number of satisfied clauses in a formula, then we cannot hope to find an approximate solution within a factor $1 - \varepsilon$ of the maximum in polynomial time, unless P = NP. (Other applications are mentioned in the *Historical Notes* section.)

Remark 2.1. — From a mathematician's perspective, the PCP theorem might appear to focus on the "uninteresting" part of mathematics. It is a restatement of NP, not of P; as such, it says nothing about the difficulty of finding proofs. Also, it treats readers as mere fact-checkers. But mathematicians read proofs not so much to find bugs in them but to understand the ideas behind them. This mental picture, so vital to mathematics, is absent from the PCP viewpoint. Within the restrictive framework of verification, the PCP theorem is an impressive statement nevertheless.

Remark 2.2. — The proof of the PCP theorem is a mix of elementary algebra and probability theory; it is long and technical but not particularly difficult. Its originality lies elsewhere: in two places to be precise. One is its use of computational self-reducibility. Instead of keeping the usual separation between proving and verifying, the verifier's work is itself re-encoded as part of the proof: the reader of a proof is made partly its author! The other intriguing aspect of the PCP theorem is its ingenious use of error-correcting codes to express not just signals and bit streams (in typical coding theory fashion) but mathematical proofs, instead.

We close this section presenting a short, archetypical motif of the proof. Given a 3-CNF formula Φ, we wish to design a PCP system to verify its satisfiability. The idea is to construct a large family of multivariate polynomials f_i such that: if Φ is satisfiable, then any satisfying truth assignment corresponds to a common zero to all the f_i's; otherwise, no more than half of them have a common zero. Suppose that the proof falsely claims that Φ is satisfiable. The verifier asks the prover to present the value of the f_i's at that common zero. If the prover obliges, then half of them will be nonzero and the verifier will easily catch the lie by random sampling. Therefore, the prover must cheat by substituting 0 for the actual values. The verifier's strategy is then to push the prover into the liar's standard pattern of generating new lies to cover up old ones.

Here is one way to do that: Encode any linear combination of the f_i's at the claimed zero as the image of a linear form g at some point x, and ask the prover to present g by values, i.e., provide a table, indexed by x, of all the values of $g(x)$. If the prover lies at one spot $g(x)$, then it must lie all over the place as well, since the verifier can evaluate $g(x)$ as $g(x+y) - g(y)$ for a random y, and hence, quickly spot any inconsistency. Many other such error-detecting mechanisms are needed. They all share the same goal, which is to force the prover to present functions that are very close to some "canonical" functions. Canonical functions are chosen to satisfy certain functional equations. Furthermore, any family of functions that satisfy these equations yield a satisfying assignment for Φ and, hence, a contradiction.

Notation. — $\mathcal{F}^d_{q,m}$ denotes the set of m-variate polynomials of total degree at most d with coefficients in $\mathbf{F}_q$, the finite field of q elements (not to be confused with the query time). We restrict ourselves exclusively to prime fields. We say that a function $f : \mathbf{F}_q^m \mapsto \mathbf{F}_q$ is δ-close to a (finite) family of functions if, for some g in that family, $\text{Prob}[f(x) \neq g(x)] \leqslant \delta$, for random $x \in \mathbf{F}_q^m$. The smallest such δ is called the distance of f to the family. Given any nonempty $H \subseteq \mathbf{F}_q$, we use $\sum_{H^m} f$ as shorthand for $\sum_{x_1,\ldots,x_m \in H} f(x_1, \ldots, x_m)$. All logarithms are to the base 2.

3. TESTING TOOLS

Intuitively, the encoding of a proof in a PCP system must be such that any local deviation from what the verifier expects should be visible nearly everywhere. The relation between a polynomial and the corresponding polynomial map shares this characteristic: Changing a polynomial map at a single point has a rippling effect visible almost everywhere. This analogy suggests a line of attack: encode proofs as polynomials. In this section we pursue this lead and build a number of algebraic tools to be used later when proving the PCP theorem. We specify a polynomial in two different ways. It can be presented, i.e., written down, *by coefficients* (with the obvious meaning) or *by values* (as mentioned earlier). The appeal of the presentation by values is that it is extremely redundant and, hence, provides a built-in error-correcting mechanism.

3.1. The Sumcheck Test

Given $f \in \mathcal{F}^d_{q,m}$, $c \in \mathbf{F}_q$, and some nonempty $H \subseteq \mathbf{F}_q$, how can a proof convince a verifier that $\sum_{H^m} f = c$? Of course, the verifier can compute the sum on its own without a proof, but that requires evaluating f at $O(|H|^m)$ points. Can it be done faster? If the field is big enough, say, $q > 2dm$, it can be done with a single evaluation of f. Write $f_a = f(a, x_2, \dots, x_m) \in \mathcal{F}^d_{q,m-1}$; if $m = 1$, $\sum_{H^{m-1}} f_a$ denotes $f(a)$. For $m > 0$, the PC proof is defined recursively as follows:

```
PC PROOF THAT $\sum_{H^m} f = c$
[1] Present $g(x) = \sum_{H^{m-1}} f_x$ by coefficients.
[2] For all $a \in \mathbf{F}_q$, write down the PC proof that $\sum_{H^{m-1}} f_a = g(a)$.
```

The recursion bottoms out at $m = 0$: no PC proof needed there. The verifier adopts a two-pronged strategy: First, trust the proof and check that it supports the claim; then, test the proof for internal consistency. Accordingly, the verifier begins by checking that $\sum_H g = c$, rejecting the proof if this fails. Next, it verifies that $g(x)$ is, indeed, the polynomial it thinks it is. It picks a random $a \in \mathbf{F}_q$, and uses the PC proof in [2] to verify recursively that, indeed, $\sum_{H^{m-1}} f_a$ is equal to $g(a)$. (If $m = 1$, the verifier does not need to go to [2], since $\sum_{H^{m-1}} f_a$ is available via a single evaluation of the polynomial f.) If this succeeds, the verifier accepts the proof and its claim that $\sum_{H^m} f = c$; else, it rejects it.

To argue that this works, we first observe that if the proof is correct, the verifier always accepts it. On the other hand, if any test fails, rejection ensues. So, the only case worth considering is where the claim is not true but all the tests pass and, therefore, the proof is accepted. Important: all subsequent correctness proofs in this paper will be limited to this case, too, without a need to repeat why.

The case $m = 0$ is error-free. Note that it is the (only) place where the verifier can match the proof against its own knowledge of f. All other tests involve only the internal consistency of the PC proof. Assume now that $m > 0$. We prove by induction on m that the verifier wrongly accepts with probability at most dm/q. Since the first test passes, $g(x)$ cannot agree with the true $\sum_{H^{m-1}} f_x$ everywhere (else their respective sums would be both equal to c). But, being univariate polynomials of degree at most d, they agree at $x = a$ with probability at most d/q. This agreement might lead the verifier to wrongly accept. If there is disagreement at $x = a$, however, the verifier is back to its old task, but now with only $m - 1$ variables. So, by induction, it is fooled with probability at most $d(m-1)/q$. Adding the earlier bound of d/q completes the induction.

LEMMA 3.1. — *Given any $f \in \mathcal{F}^d_{q,m}$, $c \in \mathbf{F}_q$, and nonempty $H \subseteq \mathbf{F}_q$, if $\sum_{x\in H^m} f(x) = c$, then there exists a proof of that fact that the verifier always accepts. Otherwise, no proof is accepted with probability greater than dm/q. The verifier reads $O(dm\log q)$ bits of the proof, needs $O(m\log q)$ random bits, and performs a single evaluation of the polynomial f.*

The most remarkable feature of this test is that only one evaluation of the function f is necessary. Note how increasing the size of the field, by making the polynomial map f increasingly redundant, has the effect of decreasing the error probability. The sumcheck test can be used in conjunction with the simple fact below to test if a function that is close enough to a polynomial vanishes everywhere.

LEMMA 3.2. — *Given $H \subseteq \mathbf{F}_q$ of size $h > 0$, it is possible to build polynomials $R_1, \ldots, R_{q^\ell}$ in $\mathcal{F}^{h\ell}_{q,\ell}$, each one in time $(h+\ell)^{O(1)}$, so that for any nonzero function $g : H^\ell \mapsto \mathbf{F}_q$ and a random index $1 \leqslant i \leqslant q^\ell$, the probability that $\sum_{H^\ell} gR_i = 0$ is at most $h\ell/q$.*

3.2. The Low-Degree Test

We wish to design a PCP system to convince a verifier that a function $f : \mathbf{F}_q^m \mapsto \mathbf{F}_q$ is close to being a polynomial. Restrictions to lines are particularly useful for that purpose. Given $t \in \mathbf{F}_q$, let $f_{a,b}(t)$ denote the univariate function $f(a+tb)$. We define $\Delta_\ell(f)$ as the expected distance of $f_{a,b}$ to $\mathcal{F}^d_{q,1}$ for random $a, b \in \mathbf{F}_q^m$.

LEMMA 3.3. — *If q/md^3 is large enough, given any function $f : \mathbf{F}_q^m \mapsto \mathbf{F}_q$, the distance of f to $\mathcal{F}^d_{q,m}$ is at most $c\Delta_\ell(f)$, for some absolute constant $c > 0$.*

Intuitively, this is saying that if the restriction of a function to a random line is close to a polynomial, so is the function itself. This suggests an obvious PCP system.

PC PROOF THAT f IS δ-CLOSE TO $\mathcal{F}^d_{q,m}$.
For every pair $a, b \in \mathbf{F}_q^m$, present by coefficients the polynomial $g_{a,b} \in \mathcal{F}^d_{q,1}$ that is closest to $f_{a,b}$.

The verifier chooses two small constant parameters $\delta, \varepsilon > 0$ and picks $k = \lceil c\delta^{-1}\log\varepsilon^{-1}\rceil$ random (a_i, b_i, t_i), with $a_i, b_i \in \mathbf{F}_q^m$ and $t_i \in \mathbf{F}_q$. Next, it checks that $g_{a_i,b_i}(t_i) = f(a_i + t_i b_i)$, for each $1 \leqslant i \leqslant k$. If all k tests succeed, then the proof is accepted. Any failure means rejection.

Correctness is immediate: If $f \in \mathcal{F}^d_{q,m}$, then $g_{a,b}(t)$ coincides with $f(a+tb)$, for all a, b, and all tests succeed. Suppose now that the distance from f to $\mathcal{F}^d_{q,m}$ exceeds δ. For any fixed a_i, b_i and random t_i, the probability that $f(a_i + t_i b_i) \neq g_{a_i,b_i}(t_i)$ is precisely the distance from f_{a_i,b_i} to $\mathcal{F}^d_{q,1}$. Averaging over all a_i, b_i, it follows from Lemma 3.3 that, for any fixed i and random a_i, b_i, t_i,

$$\text{Prob}[f(a_i + t_i b_i) \neq g_{a_i,b_i}(t_i)] = \Delta_\ell(f) > \frac{\delta}{c}\,;$$

therefore, all k tests succeed with probability less than $(1 - \delta/c)^k < \varepsilon$. This implies the following lemma.

LEMMA 3.4. — *Let δ, ε be two arbitrarily small positive constants. Given a function $f : \mathbf{F}_q^m \mapsto \mathbf{F}_q$, fix some integer d such that q/md^3 is large enough. If $f \in \mathcal{F}^d_{q,m}$, then there exists a proof that the verifier always accepts. If f is not δ-close to $\mathcal{F}^d_{q,m}$, then no proof is accepted with probability greater than ε. The verifier reads $O(d \log q)$ bits of the proof, needs $O(m \log q)$ random bits, and performs $O(1)$ evaluations of f.*

If f is δ-close to $\mathcal{F}^d_{q,m}$, then its nearest polynomial (unique for small enough δ) can be evaluated at a random point by using f. To evaluate it at an arbitrary point, however, requires a recovery mechanism. The striking feature of the result below is that a single evaluation of f is sufficient to recover $f(x)$ at k (non-necessarily random) points.

LEMMA 3.5. — *For $\delta > 0$ small enough, fix d such that $q\delta/kd$ is large enough. Given a function $f : \mathbf{F}_q^m \mapsto \mathbf{F}_q$, let f_o be a nearest polynomial in $\mathcal{F}^d_{q,m}$. Pick k arbitrary points, $z_1, \ldots, z_k$ in $\mathbf{F}_q^m$. If f is δ-close to $\mathcal{F}^d_{q,m}$, then f_o is unique and there exists a proof that allows the verifier to output $f_o(z_1), \ldots, f_o(z_k)$.*

Otherwise, with probability $1 - O(\sqrt{\delta}\,)$, the verifier either outputs the right values or rejects the proof. The verification reads $O(dk \log q)$ bits of the proof, needs $O(m \log q)$ random bits, takes $(mdk \log q)^{O(1)}$ time, and performs a single evaluation of the function f.

3.3. The Linearity Test

How hard is it to tell whether a function is almost a linear form, i.e., $x \in \mathbf{F}_q^m \mapsto a^T x$, for some $a \in \mathbf{F}_q^m$? We restrict ourselves to the case $q = 2$, the only one of interest for our purposes. Consider a function $f : \mathbf{F}_2^m \mapsto \mathbf{F}_2$ such that, for random $x, y \in \mathbf{F}_2^m$, $\text{Prob}[f(x) + f(y) \neq f(x + y)] \leqslant \delta$, for some small enough $\delta > 0$. A simple argument shows that the function is 2δ-close to some linear form. By now, we trust that the reader can easily write a PC proof for linearity testing.

LEMMA 3.6. — *Given a function $f : \mathbf{F}_2^m \mapsto \mathbf{F}_2$, fix a small enough constant δ. If f is a linear form, then there exists a proof of that fact that the verifier always accepts. On the other hand, if its distance to any linear form exceeds δ, no proof is accepted with probability greater than δ. The verifier reads $O(1)$ bits of the proof, needs $O(m)$ random bits, and performs $O(1)$ evaluations of the function f.*

4. THE PCP THEOREM

The proof consists of three parts: the first two involve the design of suboptimal PCP systems for 3-SAT; the third provides a composition method that allows us to plug the two suboptimal schemes together to produce the desired PCP system.

4.1. Optimal Random Bit Complexity: $\mathrm{NP} \subseteq \mathrm{PCP}[\log n, (\log n)^{O(1)}]$

Let Φ be a 3-CNF formula consisting of m clauses $C_1, \dots, C_n$ and n variables $v_1, \dots, v_n$. Since $m = O(n^3)$ and our PCP bounds in this section are all (poly)logarithmic, we might as well assume that $m = n$. We associate a polynomial with each clause in a fairly obvious way: $x_i(1-x_j)x_k$ with $\neg v_i \vee v_j \vee \neg v_k$; plus seven other possibilities. It is clear that Φ is satisfiable if and only if all these n polynomials have a simultaneous zero over $\mathbf{F}_q$. Given a 0/1 assignment f of the x_i's, we define a function $G^f : \{1, \dots, n\}^4 \mapsto \{0,1\}$ as follows:

$$G^f(i,j,k,l) = \begin{cases} 0 & \text{if } v_i, v_j, v_k \text{ do not all appear} \\ & \text{in } C_l \text{ in that order;} \\ f(i)f(j)f(k) & \text{if } C_l \text{ is } \neg v_i \vee \neg v_j \vee \neg v_k\,; \\ f(i)f(j)(1-f(k)) & \text{if } C_l \text{ is } \neg v_i \vee \neg v_j \vee v_k\,; \\ \text{etc.} & \text{(6 other cases).} \end{cases} \tag{2}$$

Φ is satisfiable if and only if there exists an assignment f such that the function G^f vanishes everywhere. To take advantage of the redundancy of polynomial maps mentioned earlier, we encode f and then G^f as polynomial maps.

Let $h = \Theta(\log n)$ and $m = \lceil h/\log h \rceil$, and define q to be a prime sufficiently larger than h^3m^4. Without loss of generality, assume that $n = h^m$. Fix a bijection between $\{1, \dots, n\}$ and H^m, where $H = \{0, \dots, h-1\}$; for example, write $i-1$ in base h. From now on, we regard the index i of x_i as an element $(y_1, \dots, y_m)$ of $H^m \subseteq \mathbf{F}_q^m$. The assignment f maps H^m to $\{0,1\}$ and, by Lagrange interpolation, can be extended into a map defined by a polynomial of degree $(h-1)m$ (still called f, for simplicity):

$$f(y_1, \dots, y_m) = \sum_{t_1, \dots, t_m \in H} f(t_1, \dots, t_m) \prod_{i=1}^{m} \prod_{u \in H \setminus \{t_i\}} \frac{y_i - u}{t_i - u}. \tag{3}$$

Similarly, we regard $G^f(i,j,k,l)$ as a function from H^{4m} to $\mathbf{F}_q$, which we extend into a polynomial G^f_{poly} in $\mathcal{F}^{9hm}_{q,4m}$. This is how we do it. First, we express G^f in a more unified manner:

$$G^f(i_1, i_2, i_3, l) = \prod_{s=1}^{3} a_s(i_s, l)\,(\,b_s(l) - f(i_s)\,),$$

with the obvious meaning of all these functions: s specifies one of the three literals in the clause C_l; i_s is the index (viewed as an element of H^m) of the corresponding variable x_{i_s} and $f(i_s)$ is its 0/1 assignment; $b_s(l) = 0/1$ indicates if x_{i_s} is negated, etc. The polynomial extension of G^f is defined as

$$G^f_{\mathrm{poly}}(i_1, i_2, i_3, l) = \prod_{s=1}^{3} a'_s(i_s, l)\,(\,b'_s(l) - f(i_s)\,),$$

where a'_s, b'_s are the polynomial extensions of a_s, b_s as defined in (3). It is immediate that G^f_{poly} is of degree less than $9hm$. (Note that we avoid Lagrange interpolation on G^f itself.)

PC PROOF THAT Φ IS SATISFIABLE

[1] Present $f : \mathbf{F}_q^m \mapsto \mathbf{F}_q$ by values, where f is a polynomial extension of a satisfying assigment for Φ.

[2] Write down the PC proof that f is ε-close to $\mathcal{F}^{hm}_{q,m}$. (*low-degree test*)

[3] Form all R_i's in Lemma 3.2 ($\ell = 4m$) and, for each $1 \leqslant i \leqslant q^{\ell}$, write down the PC proof that $\sum_{H^{4m}} G^f_{\mathrm{poly}} R_i = 0$. (*sumcheck test*)

The verifier applies the low-degree test on f and rejects the proof if it fails. Otherwise, it picks a random $1 \leqslant i \leqslant q^{4m}$, and sumchecks that $\sum_{H^{4m}} G^f_{\mathrm{poly}} R_i = 0$. It accepts the proof if this test succeeds and rejects it otherwise.

Remark 4.1. — The verifier can compute R_i and a'_s, b'_s on its own and also evaluate G^f_{poly} anywhere by querying f at three places in the proof. Also, we said earlier that the sumcheck test requires that the verifier can trust its evaluations of f. But what if the prover cheated in the presentation of f? This cannot happen: whatever is presented *is* what defines f.

Why does this protocol work? As usual, we do not have false negatives. If Φ is satisfiable, then the prover only has to stick to the scenario above and all tests will succeed. Suppose now that Φ cannot be satisfied. For the usual reasons, we assume that all tests (i.e., low-degree and sumcheck) succeed. We distinguish between three cases:

(1) $f \in \mathcal{F}^{hm}_{q,m}$: Then, G^f_{poly} is a polynomial of degree at most $9hm$, which is nonzero because Φ is not satisfiable. So, by Lemma 3.2, the probability that the sum to check is 0 is at most $4hm/q$. Assume that it is not 0.

The degree of $G^f_{\mathrm{poly}} R_i$ does not exceed $13hm$; therefore, by Lemma 3.1, the probability that the sumcheck test succeeds is $O(hm^2/q)$. This bounds the probability of failure in this case by $O(hm^2/q)$.

(2) f is ε-close (but not 0-close) to $\mathcal{F}^{hm}_{q,m}$: Let f_o be a nearest neighbor in $\mathcal{F}^{hm}_{q,m}$. The sumcheck test requires a single evaluation of G^f_{poly} and, hence, evaluations of f at three points. The probability that f and f_o agree at all three points is at least $1-3\varepsilon$. The agreement means that the sumcheck test is, in effect, carried out on $G^{f_o}_{\text{poly}} R_i$. The previous case now shows that the failure probability is at most $3\varepsilon + O(hm^2/q)$.

(3) f is not ε-close to $\mathcal{F}^{hm}_{q,m}$: By Lemma 3.4, the low-degree test will fail to catch that fact with probability at most ε.

To summarize, the verifier might fail to spot an inconsistency with probability $O(\varepsilon + hm^2/q) < 1/2$, for a small enough constant $\varepsilon > 0$. Since $q = O(m^4h^3)$, the number of proof bits read is $O(hm^2 \log q) = (\log n)^{O(1)}$, the running time is $(hm)^{O(1)} = (\log n)^{O(1)}$, the number of random bits needed is $O(m \log q) = O(\log n)$ (the motivation for our choice of h), and the number of places at which the candidate assignment function is evaluated is $O(1)$. This proves that, indeed, NP $\subseteq$ PCP$[\log n, (\log n)^{O(1)}]$.

4.2. Optimal Query Bit Complexity: NP $\subseteq$ PCP$[n^{O(1)}, 1]$

Adding random bits allows the verifier to do with fewer lookups; in fact, a constant number of them. The strategy is roughly the same as before. Since the verifier is given access to only $O(1)$ bits of the proof, however, the ground field must be of constant size, so we set $q = 2$.

Let Φ be a 3-CNF formula consisting of m clauses $C_1, \ldots, C_m$ and n variables $v_1, \ldots, v_n$. As usual, we model each clause, say, $\neg v_i \vee v_j \vee \neg v_k$ as $x_i(1-x_j)x_k$. This defines m cubic polynomials $G_1, \ldots, G_m$, and Φ is satisfiable if and only if all the G_i's have a common zero over $\mathbf{F}_2$. Given $r = (r_1, \ldots, r_m) \in \mathbf{F}_2^m$, let $F_r(x) = \sum r_i G_i(x)$. Our interest in F_r comes from this (trivial) fact:

LEMMA 4.2. — *If Φ is satisfiable, all 2^m polynomials F_r have a common zero; otherwise, given any $a \in \mathbf{F}_2^n$ and a random $r \in \mathbf{F}_2^m$, $F_r(a) = 0$ with probability $1/2$.*

The PC proof of satisfiability is based on this simple test. The prover wants to convince the verifier that it knows a common zero a to all the F_r's (whether that is true or not). To do that, the proof will list the values of $F_r(a)$, for all r, so that the verifier can test that, indeed, $F_r(a) = 0$. The prover must also provide a consistency check that satisfies the verifier that its evaluations of F_r are correct. The cubic polynomial F_r can be written as $F_r(x) = f_r + \sum_i f_r^i x_i + \sum_{i,j} f_r^{ij} x_i x_j + \sum_{i,j,k} f_r^{ijk} x_i x_j x_k$. The verifier can evaluate $F_r(a)$ by using the three linear forms

$$(4) \qquad H_1^a(y) = \sum_i a_i y_i\,; \qquad H_2^a(y) = \sum_{i,j} a_i a_j y_{ij}\,; \qquad H_3^a(y) = \sum_{i,j,k} a_i a_j a_k y_{ijk}\,.$$

Forgive our (abusive) notation y to refer to a set of n, n^2, and n^3 labeled variables, respectively. The PC proof will present each H_i^a by values ($i = 1, 2, 3$). Of course, no guarantee exists that the prover will not corrupt the presentation.

To catch any cheating, the verifier relies on two sets of functional equations: the H_i^a's are (i) linear and (ii) related by the identities

$$(5)\qquad H_2^a(y \otimes y') = H_1^a(y)H_1^a(y') \qquad \text{and} \qquad H_3^a(y \otimes z) = H_1^a(y)H_2^a(z)\,,$$

where $y, y' \in \mathbf{F}_2^n$ and $z \in \mathbf{F}_2^{n^2}$. If $y \in \mathbf{F}_2^s$ and $z \in \mathbf{F}_2^t$, then $y \otimes z$ denotes the vector $(y_i z_j) \in \mathbf{F}_2^{st}$.

PC Proof that Φ is Satisfiable

Present the functions H_1^a, H_2^a, H_3^a by values, where a is a common zero to $G_1, \ldots, G_m$.

The verifier performs three basic sets of tests. Fix some small constant $\varepsilon > 0$.

– The first test is to check the linearity of each H_i^a, by using the criterion of Lemma 3.6, with $\delta = \varepsilon^2$. The verifier rejects the proof if any of these 3 tests fail. From now on, *any* evaluation of $H_i^a(y)$ is to be immediately confirmed by the following test: pick a random y' and check that $H_i^a(y) = H_i^a(y + y') - H_i^a(y')$. If this test ever fails, the proof is rejected.

– Next, the verifier checks that the H_i^a's are related by the two identities (5). Each one is tested $O(1/\varepsilon)$ times for random pairs (y, y') and (y, z), where $y, y' \in \mathbf{F}_2^n$ and $z \in \mathbf{F}_2^{n^2}$. Again, any failure implies rejection of the proof.

– Finally, the verifier picks a random r and evaluates $F_r(a)$. To do that, it computes on its own $y_1 = (f_r^i)$, $y_2 = (f_r^{ij})$, $y_3 = (f_r^{ijk})$, as well as f_r, and then looks up the proof at three places to compute the sum $F_r(a) = f_r + H_1^a(y_1) + H_2^a(y_2) + H_3^a(y_3)$. If $F_r(a) = 0$ and none of the previous tests have failed, the verifier accepts the claim that Φ is satisfiable.

Why does this work? If Φ is satisfiable, then the proof needs simply to conform to the directives of the verifier and it will be accepted. Suppose that Φ is not satisfiable and that, by contradiction, the proof is accepted. What is the probability of failure? If the linearity tests passes then, by Lemma 3.6, with probability a least $1 - 3\varepsilon^2$, there exists a linear form $\widehat{H}_i$, for $i = 1, 2, 3$, that disagrees with H_i^a over a fraction at most ε^2 of its domain. This means that, with probability $1 - O(\varepsilon)$, we can assume that all identity tests are performed with respect to the true values of $\widehat{H}_i$. If either identity fails to be satisfied by the $\widehat{H}_i$'s, then by Lemma 4.3 (and its omitted analog for the first identity) a conservative estimate of $O(\varepsilon)$ bounds the probability that the verifier fails to catch that fact.

Therefore, with probability $1 - O(\varepsilon)$, the value $F_r(a)$ computed by the verifier is, indeed, $f_r + \widehat{H}_1(y_1) + \widehat{H}_2(y_2) + \widehat{H}_3(y_3)$, for some linear forms $\widehat{H}_i$ defined by some vector a in accordance with the format specified by (4). Since Φ is not satisfiable, we know by Lemma 4.2 that the value of $F_r(a)$ is zero with probability 1/2. Therefore, the verifier will accept a wrong proof with probability $1/2 + O(\varepsilon)$. By setting ε to a small enough constant and repeating the verification, we bring the failure probability below $1/2$. The number of random bits is $O(n^3)$ and the number of bit lookups in the proof is constant. This concludes the proof that $\mathrm{NP} \subseteq \mathrm{PCP}[n^{O(1)}, 1]$.

LEMMA 4.3. — *If $\widehat{H}_3(y \otimes z) - \widehat{H}_1(y)\widehat{H}_2(z)$ is nonzero then, with probability $1/4$, it evaluates to 1 at random y, z.*

4.3. Self-Reduction: $\mathrm{NP} = \mathrm{PCP}[\log n, 1]$

We have built two PCP systems: one, S_1, needs $O(\log n)$ random bits; the other, S_2, uses $O(1)$ queries. We now combine them to extract the best feature from each. The basic idea is simple. We caught a glimpse of it in Section 2. Recall the action of the verifier V_1 for S_1. First, it generates a random bit string s; then, it computes a set of addresses $i_1, \ldots, i_q$ to look up in the proof Π, where $q = (\log n)^{O(1)}$. Upon reading the corresponding bits, $\Pi_{i_1}, \ldots, \Pi_{i_q}$, the verifier evaluates a predicate in time $q^{O(1)}$ to decide whether to accept or not.

Now comes the self-reducibility part. By the Cook-Levin theorem the predicate in question can be expressed as a 3-CNF formula Φ_s of size $q^{O(1)}$. The verifier accepts if and only if there exists a string X_s such that the concatenated string $\Pi_{i_1} \cdots \Pi_{i_q} X_s$ forms a satisfying truth assignment for Φ_s. The key idea is that a PCP system such as S_2 is exactly the sort of thing that can be used to check the satisfiability of Φ_s. So, instead of computing the verification predicate itself, V_1 can hand the problem over to the PC proof system S_2. Its verifier V_2 will then consult its own proof to check whether Φ_s is satisfiable, and will accept or reject accordingly. A minor technical point: Of course, we cannot let *both* V_1 and V_2 err with probability $1/2$. By repeating the verifiers' runs (the standard trick), we can trivially lower the odds of an error to any small constant.

Even though the number of bits read in the proof can be arbitrary, it is important for composition purposes that the number of *entries* be $O(1)$. Polynomial extensions give us a convenient tool for achieving that. A proof Π of length N can be viewed as a function $f : \{1, \ldots, N\} \mapsto \{0, 1\}$, where $f(i) = \Pi_i$. As we did in Section 4.1, we can change our point of view and regard f as a function from H^m to $\{0, 1\}$, where H^m is in bijection with $\{1, \ldots, N\}$ and $H = \{0, \ldots, h-1\}$, for some parameters h, m such that $N = h^m$. (Pad the proof with junk if N is inexpressible in this way.) Let $\widehat{f}$ be a polynomial extension of f in $\mathcal{F}^{hm}_{q,m}$. The proof Π is now rewritten as a presentation of $\widehat{f}$ by values, with all the bells and whistles needed to apply the low-degree test and the recovery mechanism (Lemmas 3.4 and 3.5). The verifier applies the low-degree test to check that the presentation is δ-close to $\mathcal{F}^{hm}_{q,m}$ (for some suitably small $\delta > 0$), and rejects the proof if it is not. Otherwise, it appeals to Lemma 3.5 to evaluate f at $k\,(= q)$ points by a single evaluation of $\widehat{f}$. In this way, the verifier can gain access to $\Pi_{i_1} \cdots \Pi_{i_q}$ in $O(1)$ queries to the new proof. Note that an entry in this new proof is no longer a single bit but a field element represented as a bit string. For simplicity, we still call the new proof Π: the difference is that now $q = O(1)$.

The benefits of composing proof systems are now obvious. Let us try our hand at composing S_1 with itself, i.e., S_1 with S_2, where S_2 denotes S_1. The verifiers for S_1 and S_2 need $O(\log n)$ and $O(\log(\log n)^{O(1)})$ random bits, respectively, i.e., a total of $O(\log n)$ of them. Obviously, the number of queries remains $O(1)$. All the verification work, being now done by S_2, amounts to $(\log(\log n)^{O(1)})^{O(1)}$, i.e., $(\log\log n)^{O(1)}$. This bound can be further reduced by iterating the composition, but this is not the way to go to make it $O(1)$. For that, we take the previous system, call it S_3, and compose it with S_2. This requires $O(\log n) + (\log\log n)^{O(1)} = O(\log n)$ random bits, $O(1)$ queries and $O(1)$ amount of work. It follows that $O(1)$ bits are read in the proof, and the PCP theorem is proven.

But is it really? The task of V_1 is not *only* to check that Φ_s is satisfiable but that $\Pi_{i_1} \cdots \Pi_{i_q}$ is part of a satisfying assignment.

Here is a simple illustration of the conundrum we face. Say, a prover claims to have a satisfying assignment for $\bigwedge_i C_i$. A verifier might want to check this by picking C_i at random and verifying that the assignment makes C_i true. But suppose that, instead, it chooses to delegate that task to some other PC proof system. A second verifier will then take C_i as input and check that it is satisfiable. But any disjunction of three literals, such as C_i, is always satisfiable. What needs to be checked is not whether C_i is satisfiable on its own, but whether it is by using the assignment *specified* by S_1. Returning to V_2, its job is not to check that Φ_s is satisfiable but that there exists X_s such that $\Pi_{i_1} \cdots \Pi_{i_q} \cdot X_s$ makes Φ_s true. To resolve this *consistency* issue is key to making self-reducibility work. This is easy to do; in fact, by reducing the number of queries to $O(1)$, we have done the hardest part already.

We sketch what remains to be done.

For the verification to be delegated to V_2, of course, it is necessary to encode $\Pi_{i_1} \cdots \Pi_{i_q} X_s$ into the format $\sigma(\Pi_{i_1} \cdots \Pi_{i_q} X_s)$ that V_2 expects. It might be tempting to simply append $\sigma(\Pi_{i_1} \cdots \Pi_{i_q} X_s)$ at the end of Π, but doing so would raise the consistency problem mentioned earlier. Instead, we must effectively *replace* Π (and not just add on to it) with the encoding σ of every possible string $\Pi_{i_1} \cdots \Pi_{i_q} X_s$. But to do so would cause the same Π_{i_j} to appear in different encodings throughout the proof, again raising consistency issues. The solution is to encode each Π_{i_j} separately. Specifically we replace each Π_{i_j} by $\sigma(\Pi_{i_j})$. Likewise, we encode X_s as $\sigma(X_s)$.

This solves one problem, consistency, only to create another one. In this scheme, V_2 does not have access to $\sigma(\Pi_{i_1} \cdots \Pi_{i_q} X_s)$, which is the only encoding it can read, but to $\sigma(\Pi_{i_1}) \cdots \sigma(\Pi_{i_q})\sigma(X_s)$. Is that good enough? Instead of encoding a whole truth assignment $a_1 \cdots a_n$ via σ, suppose we encode it in chunks: first $\sigma(a_1 \cdots a_{i_1})$, then $\sigma(a_{i_1+1} \cdots a_{i_2})$, etc, and finally $\sigma(a_{i_{q-1}+1} \cdots a_n)$. Can the verifier deal with that sort of *split form* encoding? The answer is yes. It is, in fact, a rather simple exercise to modify V_2 accordingly.

This completes the proof of the PCP theorem or, at least, of its conceptual outline. The doubting reader can always sample the proof at random and see if that helps...

5. HISTORICAL NOTES

Following the seminal work of Goldwasser, Micali, and Rackoff [20] and Babai [5], which introduced the notion of interactive proofs, an important variant was introduced by Ben-Or et al. [9], in which the verifier interacts with not one but several provers. This framework led to early incarnations of PCP systems by Fortnow, Rompel and Sipser [14]. The idea of putting tight resource restrictions on both the verifier (query time) and the proof (size) originated in the works of Babai et al. [6] and Feige et al. [13]. The algebraic view of Boolean expressions gained currency in a series of papers that highlighted the enormous expressive power of interactive proofs [7, 24]. Turning to co-NP, Lund et al. [24] explained how a prover can convince a verifier that a graph is *not* 3-colorable. (By contrast, to convince someone that a graph is 3-colorable is trivial.) Finally, the ultimate power of interaction was resolved by Shamir [30], who proved that languages with interactive proofs are precisely those that can be decided in polynomial space.

The current notion of PCP itself, with its focus on randomness and query complexity, was formally introduced by Arora and Safra [4]. This development was spurred in large measure by the key insight of Feige et al. [13], which for the first time tied probabilistic proof systems to inapproximability. Babai, Fortnow, and Lund [7] established that $\mathrm{PCP}[n^{O(1)}, n^{O(1)}]$ coincides with the class of problems solvable in nondeterministic exponential time. Babai et al. [6] and Feige et al. [13] essentially showed that NP is contained in PCP[polylog,polylog] (the precise bounds being somewhat stronger). Arora and Safra [4] proved that $\mathrm{NP} = \mathrm{PCP}[\log n, \sqrt{\log n}]$, and introduced the powerful concept of proof composition. The PCP theorem itself, i.e., $\mathrm{NP} = \mathrm{PCP}[\log n, 1]$, was proven by Arora et al. [3]. Finetuning the constants followed in quick order. Håstad [22] proved the striking result that three queries are sufficient as long as we can tolerate an ε chance of rejecting a correct proof. Building on that result, Guruswami et al. [21] showed that such false-negatives can be avoided provided that the error probability for wrongly accepting is $1/2 + \varepsilon$.

The connection to inapproximability [13] blossomed into a plethora of hardness results, one of the most impressive being Håstad's proof [23] that to approximate the clique number of an n-node graph within a factor of $n^{1-\varepsilon}$ is impossible (unless NP coincides with the randomized version of P, i.e., the class of sets for which membership can be decided in expected polynomial time by a randomized, error-free Turing machine). At the other end of the spectrum, consider MaxCut, the problem of partitioning the node set of a graph into two subsets with the maximum number of edges joining them. It is possible to find a solution in polynomial time that has a number of edges at least 0.878 times the maximum possible [17]. On the other hand, to push that approximation factor above 0.942 would require that P = NP [22, 33] (building on [8]). A comprehensive 1996 survey of approximation results was compiled by Arora and Lund [2].

Many of the tools for checking the internal consistency of proof systems originated in the area of *program checking* [10, 11, 29]. For example, the low degree test, due to Arora et al. [3], incorporates ideas from [4, 11, 16, 28].

The sumcheck and linearity tests are due respectively to Lund et al. [24] and Blum, Luby, and Rubinfeld [11]. Testing that a polynomial is nonzero (Lemma 3.2) is from [6, 13].

Essential tools in PCP-related work also include the *parallel repetition theorem* by Raz [27], the *long code* by Bellare, Goldreich and Sudan [8], and Fourier transform techniques by Håstad [22]. For background material in complexity theory, the following texts [15, 31, 26, 12], listed in increasing order of technical depth, are excellent entry points. We also mention [1, 2, 25, 32] for in-depth coverage of proof verification and approximation algorithms, and [18, 19, 20] for an introduction to zero-knowledge cryptography.

Acknowledgments. — I wish to thank Lance Fortnow, Oded Goldreich, and Muli Safra for their helpful comments on this manuscript.

REFERENCES

[1] S. Arora – Probabilistic Checking of Proofs and the Hardness of Approximation Problems, Ph.D. Thesis, UC Berkeley, 1994, also available as `http://www.cs.princeton.edu/~arora`.

[2] S. Arora & C. Lund – Hardness of approximations, in *Approximation Algorithms for NP-hard Problems* (D. Hochbaum, ed.), PWS Publishing, 1996.

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

[4] S. Arora & M. Safra – Probabilistic checking of proofs: a new characterization of NP, *J. ACM* **45** (1998), p. 70–122.

[5] L. Babai – Trading group theory for randomness, in *Proc. 17th Annual ACM, Symp. Theory Comput.*, 1985, p. 421–429.

[6] L. Babai, L. Fortnow, L. Levin & M. Szegedy – Checking computations in polylogarithmic time, in *Proc. 23rd Annual ACM Symp. Theory Comput.*, 1991, p. 21–31.

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

[8] M. Bellare, O. Goldreich & M. Sudan – Free bits, PCPs and non-approximability—towards tight results, *SIAM J. Comput.* **27** (1998), p. 804–915.

[9] M. Ben-Or, S. Goldwasser, J. Kilian & A. Wigderson – Multi-prover interactive proofs: how to remove intractability, in *Proc. 20th Annual ACM Symp. Theory Comput.*, 1988, p. 113–131.

[10] M. Blum & S. Kannan – Designing programs that check their work, in *Proc. 21st Annual ACM Symp. Theory Comput.*, 1989, p. 86–97.

[11] M. Blum, M. Luby & R. Rubinfeld – Self-testing/correcting with applications to numerical problems, *J. Comp. Sys. Sci.* **47** (1993), p. 549–595.

[12] D.-Z. Du & K.-I. Ko – *Theory of Computational Complexity*, Wiley-Interscience, 2000.

[13] U. Feige, S. Goldwasser, L. Lovasz, S. Safra & M. Szegedy – Interactive proofs and the hardness of approximating cliques, *J. ACM* **43** (1996), p. 268–292.

[14] L. Fortnow, J. Rompel & M. Sipser – On the power of multi-prover interactive protocols, *Theoret. Comput. Sci.* **134** (1994), p. 545–557.

[15] M. Garey & D. Johnson – *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[16] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan & A. Wigderson – Self-testing/correcting for polynomials and approximate functions, in *Proc. 23rd Annual ACM Symp. Theory Comput.*, 1991, p. 32–42.

[17] M.X. Goemans & D.P. Williamson – Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *J. ACM* **42** (1995), p. 1115–1145.

[18] O. Goldreich – *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Algorithms and Combinatorics, vol. 17, Springer, 1999.

[19] O. Goldreich, S. Micali & A. Wigderson – Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems, *J. ACM* **38** (1991), p. 691–729.

[20] S. Goldwasser, S. Micali & C. Rackoff – The knowledge complexity of interactive proof systems, *SIAM J. Comput.* **18** (1989), p. 186–208.

[21] V. Guruswami, D. Lewin, M. Sudan & L. Trevisan – A tight characterization of NP with 3 query PCPs, in *Proc. 39th Annual IEEE Symp. Found. Comput. Sci.*, 1998, also available as ECCC Technical Report TR98-034, p. 8–17.

[22] J. Håstad – Some optimal inapproximability results, in *Proc. 29th Annual ACM Symp. Theory Comput.*, 1997, also available as ECCC Technical Report TR97-037, p. 1–10.

[23] ———, Clique is hard to approximate within $n^{1-\varepsilon}$, *Acta Mathematica* **182** (1999), p. 105–142.

[24] C. Lund, L. Fortnow, H. Karloff & N. Nisan – Algebraic methods for interactive proof systems, *J. ACM* **39** (1992), p. 859–868.

[25] E.W. Mayr, H.J. Promel & A. Steger – *Lectures on Proof Verification and Approximation Algorithms*, LNCS, vol. 1367, Springer Verlag, 1998.

[26] C.H. Papadimitriou – *Computational Complexity*, Addison Wesley, 1994.

[27] R. Raz – A parallel repetition theorem, *SIAM J. Comput.* **27** (1998), p. 763–803.

[28] R. Rubinfeld & M. Sudan – Self-testing polynomial functions efficiently and over rational Domains, in *Proc. 3rd Annual ACM/SIAM Symp. Discrete Algorithms*, 1992, p. 23–32.

[29] ______, Robust characterizations of polynomials with applications to program testing, *SIAM J. Comput.* **25** (1996), p. 252–271.

[30] A. Shamir – IP = PSPACE, *J. ACM* **39** (1992), p. 869–877.

[31] M. Sipser – *Introduction to the Theory of Computation*, PWS Publishing, 1997.

[32] M. Sudan – Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems, in *ACM Distinguished Dissertation Series for 1993*, Springer, 1996.

[33] L. Trevisan, G.B. Sorkin, M. Sudan & D.P. Williamson – Gadgets, Approximation, and Linear Programming, *SIAM J. Comput.* **29** (2000), p. 2074–2097.

Bernard CHAZELLE

Princeton University
Department of Computer Science
Princeton, NJ 08544
USA
E-mail : chazelle@cs.princeton.edu