SÉMINAIRE N. BOURBAKI

BERNARD CHAZELLE

The PCP theorem

Séminaire N. Bourbaki, 2001-2002, exp. n° 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 $L \in \text{NP}$ 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 $\text{PCP}[r(n), q(n)]$:

– 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 $\text{PCP}[\log n, 1] \subseteq \text{NP}$. 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

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

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