Electronic Colloquium on Computational Complexity, Report No. 27 (2001)

Probabilistically Checkable Proofs The Easy Way

Marius Zimand [*]

Abstract

We present a weaker variant of the PCP Theorem that admits a significantly easier proof. In this variant the prover only has n^t time to compute each bit of his answer, for an arbitray but fixed constant t, in contrast to being all powerful. We show that 3SAT is accepted by a polynomial-time probabilistic verifier that queries a constant number of bits from a polynomially long proof string. If a boolean formula ϕ of length n is satisfiable, then the verifier accepts with probability 1. If ϕ is not satisfiable, then the probability that a n^t-bounded prover can fool the verifier is at most 1/2. The main technical tools used in the proof are the "easy" part of the PCP Theorem in which the verifier reads a constant number of bits from an exponentially long proof string, and the construction of a pseudo-random generator from a one-way permutation.

1 Introduction

The PCP Theorem is one of the most important results in computer science. It gives an astonishing interactive proof protocol for any language in NP and it has been used to obtain impressive lower bounds on the approximation ratio that can be achieved in polynomial time for many important optimization combinatorial problems. The theorem states that for any language L in NP, a polynomial-time probabilistic verifier needs to check only a constant number of bits of a polynomially long proof of membership; if an input x is in L, then the verifier accepts with probability one (completeness condition), and if x is not in L, the verifier can be fooled to accept a (false) proof of membership with probability at most, say, 1/4 (the soundness condition).

The proof of the PCP Theorem is long and difficult. Some researchers (Goldreich [Gol99, pp.71], Sudan [Sud00b], Trevisan [Tre00]) have formulated as an open problem the discovery of a simpler proof. Trevisan [Tre00] says that a respectable task is to "focus on statements which (i) can be derived from (but are weaker than) the PCP Theorem, (ii) are already surprising enough to be interesting, (iii) are not known to have simple proofs; and to try and find simple proofs for such statements." Why would a weaker version of the PCP Theorem having a simpler proof be worthy of interest?

One reason may be practical. A direct utilization of the PCP Theorem would be in program verification, that is in building certificates for the output of a program that can be checked by reading only a constant number of bits from them. The complexity of the construction in the proof of the PCP Theorem has deterred such an application. A simpler construction, even with weaker guarantees, may be useful here. Other possibilities are suggested in the final paragraph of this section.

Another reason is pedagogical. A book entitled "PCP for Dummies" would be an immediate best-seller. Joke aside, given its stunning content, as well as its importance, it would be desirable to present the PCP Theorem in any course on computability or computational complexity at the graduate level (or even below). In a theory course, presenting means proving. There have been courses that have presented a proof of the PCP Theorem (see, for example, Sgall [Sga98] or Sudan [Sud00b]) but, to our knowledge, these courses were at an advanced level and have dedicated their entire time (or a big fraction of it) to this

[*]Department of Computer and Information Sciences, Towson University, Baltimore, MD, and Department of Computer Science, University of Bucharest, Bucharest, Romania.

ISSN 1433-8092

goal. A weaker variant of the PCP Theorem that retains some of its striking characteristics, and whose proof can be covered in, say, 2-3 lectures would be, we think, valuable for standard Theory courses.

In this paper we present a weaker variant of the PCP theorem (the PCP Light Theorem) that is of interest because of the above reasons. In this variant, the verifier is still a polynomial-time probabilistic machine. The verifier still reads a constant number of bits from a membership proof that is polynomially long. The variant still applies to languages L in NP. The completeness condition is the same. The weakening is in the soundness condition. We will show that the protocol is safe against (dishonest) provers that can spend at most polynomial-time computational power to calculate each bit of the proof string (in the PCP theorem the dishonest provers can have unlimited power). Such provers are called *scribes.* The soundness condition says that if the input string x is not in L, no scribe can fool the verifier to accept x but with constant probability. More precisely, a scribe is a polynomial-size circuit that has as input (1) the string x the verifier wants to check if it is in L or not, (2) some other string of polynomial size in $|x|$ that may be the witness for the membership of x in L, and (3) a natural number i. On this input the scribe produces the i-th bit of the proof string. Thus in case x is in L, a scribe will be given a witness to show this and will produce a proof string to convince the verifier that x is in L. Note that even though polynomially bounded, the computational power of the scribes can be much higher than the computational power of the verifier.

The merit of this result is that it has a simpler proof whose structure we sketch here. The proof has two parts. The first part is taken from the proof of the full PCP Theorem. Namely, it is the "easy" step which states that 3SAT $\in \mathrm{PCP}_{1,1/4}(O(n^3), O(1))$. This means that for a boolean formula ϕ in 3CNF, having length $|\phi| = n$, the verifier is using $O(n^3)$ random bits and reads $O(1)$ bits from a string w provided by the prover. If ϕ is satisfiable, then there is a string w (the correct proof string) so that the verifier accepts with probability 1. If ϕ is not satisfiable, then for any string w, the verifier accepts with probability at most $1/4$. The proof of this step uses the main tools of the full PCP proof (i.e., arithmetization, error-correcting codes, consistency tests) in an elegant and relatively easy to understand way. However, the number of random bits is $poly(n)$ and, therefore, the length of the proof string w is exponential.

The second part reduces the length of the proof string w to $poly(n)$. The idea is to use sampling. The verifier selects a random subset $A \subset B$ of polynomial size, where B is the set of all $2^{O(n^3)}$ strings that can be chosen as random strings in the $\mathrm{PCP}_{1,1/4}(O(n^3), O(1))$ protocol. The verifier will always choose a random string from A, identified by its rank in A, and thus the number of random bits is $O(\log|A|) = O(\log n)$. If the prover does not know A, this does not reduce the length of the proof string, because the proof string must still contain the responses to all the queries that can be calculated by the verifier with the random strings chosen from the entire B. If the prover knows A, then the proof string can be of polynomial length (because the prover can prepare the answers to only those queries produced from the random strings in the sample set A), but, in this case, normal sampling is no longer guaranteed to give an accurate estimate of the probability in the $\mathrm{PCP}_{1,1/4}(O(n^3), O(1))$ protocol. Indeed, a dishonest prover, knowing the sample points in A, could provide some answers that lead the verifier to inadvertently accept with a probability much larger than in the case the random string is chosen from B. We need to produce sample points (i.e., the elements of the set A) that are good for estimating the average value of a function, even if the function is chosen afterwards and can depend on the sample points. In some circumstances this is possible: we show, roughly speaking, that a modified sampling procedure continues to be accurate if the prover knows A, provided that $|A| = n^{(4+\epsilon)t}$ and that the function that is sampled is computable by a circuit with oracle access to A and of size n^t, for an arbitrary t. The modified sampling procedure, dubbed "sampling under adverse conditions," has been introduced by us in [Zim99] but we present here a simpler proof. "Sampling under adverse conditions" relies on the construction of a pseudo-random generator from a one-way function [HILL99]. Fortunately, what we need here is the easy case of that construction, when the one-way function is a permutation, and thus the whole proof remains relatively simple.

Interactive proof systems in which the prover is computationally bounded fave been considered before. Argument systems have been introduced by Brassard, Chaum, and Crépeau [BCC88], and they require that the prover has access to an auxiliary input (as our scribe does) and that it runs in probabilistic polynomial-time. Computationally-Sound (CS) Proofs have been introduced by Micali [Mic94] to handle problems beyond NP. CS-proofs require that the prover writes down a proof in time polynomial in the decision time, and that the verifier works in time polynomial in the input length and polylog in the decision procedure (for example, for a language in EXP, the verifier works in polynomial time). Argument systems have been used to reduce the communication complexity of the interaction between the prover and the verifier. Kilian [Kil92] has shown that under a complexity assumption (existence of strong collision-free functions), for any $L \in$ NP there is an argument system with communication complexity $polylog(n)$. CS-proofs have been used to reduce the work of the verifier (for languages above EXP). Micali [Mic94] has shown that CS-proofs exist for any recursive language if the prover and the verifier have access to a random oracle. The proofs in both [Kil92] and [Mic94] use the PCP Theorem to produce a "holographic" proof and then use cryptographic techniques (that need the assumption that the prover is computationally bounded) to shrink the proof string. It may be possible that the PCP Light Theorem can be used in the first step of these constructions rendering them more practical but further investigation is needed in this direction.

2 The model

Let us first recall the standard model of PCP$[r(n), q(n)]$. A verifier V executing a PCP$[r(n), q(n)]$ protocol is a polynomial-time probabilistic Turing machine that in addition to its working tapes has three special tapes:

- the input tape, containing the input string x having length n,
- the random tape, containing the random bits forming a string ρ of length $r(n)$ that the verifier will use in its computation, and
- the proof tape, that contains the proof string w.

The verifier based on the input x and on ρ, first determines $q(n)$ bit positions in the proof string that it wants to read, reads these bits, and then performs some additional polynomial-time calculation at the end of which it accepts or it rejects the input.

Let the output of the above computation of the verifier be denoted by $V(\rho, w, x)$. A language L is in PCP$_{c(n),s(n)}[r(n), q(n)]$ if there is a verifier V executing a PCP$[r(n), q(n)]$ protocol with the following properties:

(i) If $x \in L$, then there is a proof string w such that $Prob_\rho(V(\rho, w, x) = \text{``}accept\text{''}) \geq c(n)$, (completeness condition)

(ii) If $x \notin L$, then for any proof string w it holds that $Prob_\rho(V(\rho, w, x) = \text{``}accept\text{''}) \leq s(n)$ (soundness condition)

Let us introduce our model called *probabilistically checkable proof with scribes*, abbreviated PCPS. For brevity, we consider the language 3SAT, but the model can be easily extended to any language in NP. A *scribe* of complexity $t(n)$ is an oracle circuit of size $t(n)$. A scribe has as input a boolean formula ϕ, an assignment for it called a, and an integer i. The scribe produces the i-th bit of the proof string. A verifier V is the same as above except that it has an extra tape called the oracle tape. A PCPS$[r(n), q(n), t(n)]$ protocol on input a formula ϕ in 3CNF of length n runs as follows:

Round 1: The verifier writes on the oracle tape a random string R of polynomial size. The string R is called the public random string because the scribe has access to it.

Round 2: The verifier produces a random string r of length $r(n)$ that it keeps private. Based on ϕ, R, and r, the verifier selects $q(n)$ addresses in the proof string (that will be provided by the scribe in Round 3). The bits of the proof strings at these addresses will be queried in Step 4.

Round 3: A scribe of complexity $t(n)$ is using an assignment for ϕ and the string R as the oracle. The scribe produces bit by bit a proof string denoted $w^R(\phi, a)$ which is passed to the verifier.

Round 4: The verifier V reads from $w^R(\phi, a)$ the bits at the addresses selected at Round 2. At the end it accepts or it rejects the input.

We denote acceptance by 1, and rejection by 0, and we denote the output of the entire protocol by $V^R(r, w^R(\phi, a), \phi)$.

The differences between a PCP and a PCPS protocol are: (a) the introduction of Round 1 in the PCPS protocol, which basically is used by the verifier to announce the prover the subset A of B as discussed in the Introduction, and (b) the fact that in the PCPS protocol, the prover (called a scribe), after being given an assignment, has limited resources to produce each bit of the proof string, while in the PCP protocol, the provers have unlimited computational power. We can consider that behind the scene there is an all-powerful prover that passes n bits of information (encoded as an assignment) to the scribe to help him make the verifier accept the formula ϕ.

Definition 2.1 *A language L is in* $\mathrm{PCPS}[r(n), q(n), t(n)]$ *if there is a verifier running a* $\mathrm{PCPS}[r(n), q(n), t(n)]$ *protocol with the following properties:*

(1) If $x \in L$, then there is a scribe of complexity $t(n)$ and a string a of length poly$(|x|)$ such that for any R

$$Prob_r(V^R(r, w^R(x, a), x) = accept) = 1.$$

(2) If $x \notin L$, then with high probability of R, for any string a and for any scribe of complexity $t(n)$, it holds that

$$Prob_r(V^R(r, w^R(x, a), x) = accept) \leq 1/2.$$

3 Main result

The weaker version of the PCP Theorem that we present is the following fact.

Theorem 3.1 *(PCP Light Theorem) For any $t \geq 3$,* $3\mathrm{SAT} \in \mathrm{PCPS}[O(\log n), O(1), n^t]$.

This fact is an immediate consequence of the PCP Theorem (and of its proof), but we will show it without using the PCP theorem. Let us first clarify the meaning of Theorem 3.1. It shows that there is a probabilistic polynomial-time machine that on input a formula ϕ of length n, does the following. It uses two random strings, R of length $poly(n)$ and r of size $O(\log n)$. It sends R to the scribe. The scribe having access to R, to ϕ, and to an assignment a for ϕ, writes the proof string, spending time at most n^t in calculating each bit of the proof string. Next the verifier reads a constant number of bits from $w^R(\phi, a)$, and accepts or rejects. If ϕ is satisfiable, then there is a scribe of complexity n^t that for any R will determine V to accept with probability 1. On the other hand, if ϕ is not satisfiable, then with high probability of R, no scribe of complexity n^t can determine V to accept except with probability of r at most $1/2$. It is noteworthy that the proof of Theorem 3.1 yields a stronger result in the sense that the verifier does not have to produce a random string R at Round 1 for each input. One public random string R is with high probability safe against all n^t scribes (with t fixed), on all strings of length n.

The proof of Theorem 3.1 relies on the following two theorems.

Theorem 3.2 $3SAT \in \mathrm{PCP}_{1,1/4}(O(n^3), O(1))$.

This is a well-known and relatively "easy" step in the proof of the PCP Theorem.

The second theorem that we use states that sampling is accurate even if the function that is sampled is chosen adversarially after the sample points have been selected, provided that the function is computable by a polynomial-size circuit.

Theorem 3.3 *For any $\alpha, \beta \geq 0$, for any τ sufficiently large, there is a function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$, and a polynomial p, such that for any natural m*

(a) For R with $|R| = p(m)$ and for r with $|r| = (4+\alpha)\cdot\tau\cdot\log m$, $f(R,r)$ has length m;

(b) With probability of R at least $1 - 2^{-poly(m)}$, for any oracle circuit C with inputs of length m, having size m^τ, and that outputs 1 or 0,

$$\left| \frac{1}{2^{|r|}} \sum_r C^R(f(R,r)) - \frac{1}{2^m} \sum_z C^R(z) \right| \leq m^{-\beta} \tag{1}$$

where the first sum is taken over all the strings r of length $(4+\alpha)\cdot\tau\cdot\log m$, and the second sum is over all the strings of length m,

(c) every bit of $f(R,r)$ can be computed in time polylog(m) independently of the other bits.

The result has been established by us in [Zim99]. In Section 4 we present a simpler proof.

Proof of Theorem 3.1. Let us fix a boolean formula ϕ in 3CNF, and let n be the length of ϕ. According to Theorem 3.2, there is a verifier V' running a $\mathrm{PCP}_{1,1/4}(cn^3, q)$ protocol for some constants c and q. The computation of V' depends on the formula ϕ, the random string ρ, and the proof string w.

We now build a verifier V that simulates V', but runs a PCPS protocol with $O(\log n)$ private random bits. One obvious problem for the simulation is that V' is using cn^3 random bits that are not disclosed to the prover, while V can only use $O(\log n)$ private random bits. To solve this problem, we consider the function f promised by Theorem 3.3 with $m = cn^3, \alpha = 1, \beta = 1$ and a constant value of τ which will be specified later. In the first round, V writes a random string R of size $p(m)$ (p is the polynomial from Theorem 3.3), with the intention of using as the random string of length cn^3 needed in the simulation of V' only strings from the set $X_R = \{f(R,r) \mid |r| = 5\cdot\tau\cdot\log m\}$. Let ℓ denote $5\cdot\tau\cdot\log m$. In the second round, V selects uniformly at random a string r in $\{0,1\}^\ell$, calculates $f(R,r)$, simulates V' with the random string $f(R,r)$ having the desired length of $m = cn^3$ to determine the addresses in the proof string that V' is going to query later. In the third round, a scribe having ϕ and an assignment a for ϕ, and having access to R, calculates a proof string $w^R(\phi,a)$ spending no more than n^t time per bit of the proof string. It then passes $w^R(\phi,a)$ to the verifier. In the fourth round, V simulates V' with the queries established in round 2, and with the proof string $w^R(\phi,a)$. If some queried addresses are not in $w^R(\phi,a)$, then V rejects. Otherwise the simulation can be completed and V gives the verdict (1 for accept, 0 for reject) that the simulated V' gives.

Let us assume that ϕ is satisfiable and a is an assignment that makes ϕ to be true. According to Theorem 3.2, there is a proof string w such that $Prob_\rho(V'(\rho,w,\phi) = 1) = 1$. An inspection of the proof of Theorem 3.2 shows that given a, each bit of w can be calculated in $O(n^3)$ time. For any R written by V in the first round, a scribe, having the assignment a, produces the bits of w that V' queries when the random string ρ is taken from X_R. It follows that for every R, and for every r of length ℓ,

$$V^R(r, w^R(\phi,a), \phi) = V'(f(R,r), w, \phi) = 1.$$

Thus, if ϕ is satisfiable, for any string R,

$$Prob_{r\in\{0,1\}^\ell}(V^R(r, w^R(\phi,a), \phi) = 1) = 1.$$

This proves the completeness condition for the PCPS protocol.

Let us consider the case in which ϕ is not satisfiable. Then for any proof string w, $Prob_\rho(V'(\rho, w, \phi) = 1) \leq 1/4$. Let us fix an assignment a for ϕ and a scribe S producing each bit of the proof string in time n^t. The scribe S, on input ϕ and a, and with some R on the oracle tape, tries to convince V to accept. We build next an oracle circuit C that simulates the whole protocol run by the verifier V' and the scribe S. The circuit C has ϕ and a embedded into its circuitry and has as input a string ρ of size cn^3. C first simulates V' and determines the addresses $i_1, \ldots, i_q$ in the proof string that are queried by V' on input formula ϕ and random string ρ. Next it simulates the scribe S to determine what are the bits at addresses $i_1, \ldots, i_q$ of the proof string $w^R(\phi, a)$ that is produced by S. Finally, the circuit C simulates the last round of the computation of V' and accepts or rejects accordingly. Thus the circuit C simulates V' on the following input: the boolean formula ϕ, the random string ρ, and the bits of the proof string obtained as specified above. C also simulates the scribe S to determine the q bits of the proof string queried by V. The simulation of V takes $p_1(n)$ for some polynomial p_1, and S produces one bit in time n^t. Thus the size of the circuit C is bounded by $p_1(n) \log p_1(n) + qn^t \leq (cn^3)^\tau = m^\tau$ for some constant τ. This is the value of τ for which we use Theorem 3.3. We denote by $\mathcal{R}$ the set of strings R for which the equation (1) holds. Recall that the size of $\mathcal{R}$ represents a fraction of $1 - 2^{-poly(m)}$ from the set of strings of length $p(m)$.

Let $g^R(\rho)$ be the output of C on input ρ running with oracle R. From the simulation it holds that for every R,

$$g^R(\rho) = V'(\rho, w^R(\phi, a), \phi).$$

Also, g^R is calculated by the oracle circuit C of size n^τ. Thus, by Theorem 3.3, if $R \in \mathcal{R}$

$$\left| Prob_{r \in \{0,1\}^\ell}(g^R(f(R,r)) = 1) - Prob_{\rho \in \{0,1\}^m}(g^R(\rho) = 1) \right| \leq m^{-1} = \frac{1}{cn^3}.$$

Since

$$Prob_{\rho \in \{0,1\}^m}(g^R(\rho) = 1) = Prob_{\rho \in \{0,1\}^m}(V'(\rho, w^R(\phi, a), \phi) = 1) \leq \frac{1}{4},$$

it follows that for $R \in \mathcal{R}$,

$$Prob_{r \in \{0,1\}^\ell}(g^R(f(R,r)) = 1) \leq \frac{1}{4} + \frac{1}{cn^3} < \frac{1}{2}.$$

Note that $g^R(f(R,r))$ is exactly $V^R(r, w^R(\phi, a), \phi)$. Therefore, for any $R \in \mathcal{R}$, if $w^R(\phi, a)$ is produced by a scribe of complexity n^t, and a is any assignment for ϕ,

$$Prob_{r \in \{0,1\}^\ell}(V^R(r, w^R(\phi, a), \phi) < 1/2.$$

This proves the soundness condition for the PCPS protocol run by V. ▮

4 Sampling under adverse conditions

We now turn to Theorem 3.3. Note that Equation (1) means that, with high probability of R, the function $f(R, \cdot)$, which is constructed in the theorem, is a pseudo-random generator in the sense that no circuit oracle C of size m^τ can distinguish between the output of $f(R, \cdot)$ and a random string of length m with a bias larger than $m^{-\beta}$. Therefore, we need to build a function depending on a string R, that with high probability of R is a pseudo-random generator. This is done by a randomized construction of a one-way permutation and by using the standard transformation of a one-way permutation into a pseudo-random generator. The construction has four steps. In Step 1, we show that a random permutation from $\{0,1\}^n$ into $\{0,1\}^n$ is a one-way permutation. This result has been obtained by Gennaro and Trevisan [GT00], but we give here a different proof, which uses a technique of Impagliazzo [Imp96], and which allows more flexibility in the choice of the parameters. (We note however that the result from [GT00] would have been sufficient here.) In the second step, using the well-known construction of Goldreich and Levin [GL89], we obtain a hidden bit, which, when appended to the one-way function from Step 1, yields a pseudo-random generator with expansion 1. In Step 3, using the well-known hybrid technique (see for example [Gol93]), we make the pseudo-random generator to produce an output with length double the length of the input. Finally, in Step 4, we use the technique of Goldreich, Goldwasser and Micali [GGM86] to obtain a pseudo-random generator with exponential expansion. The constructions in Steps 2, 3, and 4, are well-known and therefore we will not present here the underlying proofs. For our claim for the relative simplicity and the pedagogical virtues of Theorem 3.1, it is however important to note that these proofs are reasonably short, self-contained, and important in their own right.

Proof of Theorem 3.3 Let n be a natural number considered as a parameter. (This is not the n from the proof of Theorem 3.1; actually it is big-O of the log of that n.)

Step 1: Build a one-way permutation. We start by taking uniformly at random a permutation $h : \{0,1\}^n \to \{0,1\}^n$.

Proposition 4.1 *Let a and b be positive real numbers and let $s = 2^{an}, t = 2^{bn}$. Let C be an oracle circuit that on inputs of length n, makes at most s queries to the oracle. Let h be a random permutation, $h : \{0,1\}^n \to \{0,1\}^n$. Then with probability of h at least $1 - 2^{-t}$, $Prob_x(C^h(x) = h^{-1}(x)) < 2e \cdot 2^{-(1-a-b)n}$.*

Proof. Let $T = \{y_1, \dots, y_t\} \subseteq \{0,1\}^n$ be a fixed set of size t. W.l.o.g., we can assume that the circuit C on an input y queries at some point its output x to check if $h(x) = y$. Let Q be the set of queries that C makes on inputs $y_1, \dots, y_t$. Clearly the size of Q, denoted $|Q|$, is at most st. The probability that C inverts $y_1, \dots, y_t$ is bounded from above by the probability that t queries from Q map via h respectively into $y_1, \dots, y_t$. The probability that t fixed queries from Q map in order into $y_1, \dots, y_t$ is $1/(N(N-1)\dots(N-t+1))$, where $N = 2^n$, and the number of ordered t-tuples chosen in Q is $|Q|(|Q|-1)\dots(|Q|-t+1) \leq st(st-1)\dots(st-t+1)$. Thus the probability that C inverts T is at most

$$\frac{st(st-1)\dots(st-t+1)}{N(N-1)\dots(N-t+1)} = \frac{\binom{st}{t}}{\binom{N}{t}} \leq \frac{(e \cdot s)^t}{\binom{N}{t}}.$$

There are $\binom{N}{t}$ ways to choose the set T in $\{0,1\}^n$, and, therefore, the expected number of sets of size t, which we denote by μ, that are inverted is at most

$$\binom{N}{t} \cdot \frac{(e \cdot s)^t}{\binom{N}{t}} = (e \cdot s)^t.$$

We take $u = 2e \cdot s \cdot t$. If there is a set of size u that is inverted, then all its subsets of size t are inverted, and there are $\binom{u}{t}$ such subsets. We have that

$$\binom{u}{t} \geq \left(\frac{u}{t}\right)^t = 2^t \cdot (e \cdot s)^t \overset{\text{def.}}{=} k.$$

By Markov Inequality, the probability that k sets of size t are inverted is at most $\mu/k \leq (e \cdot s)^t/k = 2^{-t}$. Thus, we have shown that the probability over h that C^h inverts $2e \cdot s \cdot t = 2e \cdot 2^{(a+b)n}$ strings of length n is at most 2^{-t}. ∎

Corollary 4.2 *Let $\gamma_1 > 0$. With probability of h at least $1 - 2^{-2^{\Omega(n)}}$, for any oracle circuit C of size at most $2^{(\frac{1}{2}-\gamma_1)n}$,*

$$Prob_{x \in \{0,1\}^n}(C^h(x) = h^{-1}(x)) \leq 2^{-\gamma_1 n}. \tag{2}$$

Proof. In Proposition 4.1, we take $t = 2^{\frac{1}{2}n}$, $s = 2^{(\frac{1}{2}-\gamma_1)n}$. Note that there are at most $(4s^2)^s$ circuits of size at most s and any such circuit can make at most s queries. It follows that the fraction of permutations h for which relation (2) does not hold is at most $(4s^2)^s \cdot 2^{-t} \leq 2^{-2^{\Omega(n)}}$. ▮

From now on we will consider only permutations h for which relation (2) holds. We call such permutations "good."

Step 2: Add one hidden bit. We take x and s two strings of length n which will be viewed as n-vectors over the field $\mathbb{Z}_2$. By a well-known result of Goldreich and Levin [GL89], the function $b(x, s) = x \cdot s$ (inner product in $\mathbb{Z}_2$) provides a so-called hidden bit for a one-way permutation h. Formally, this means that for any "good" permutation h, for any oracle circuit C of size $2^{(\frac{1}{2}-\frac{5}{3}\gamma_1)n}$,

$$Prob_{x,s}(C^h(h(x), s) = b(x, s)) \leq \frac{1}{2} + \frac{1}{2^{\gamma_2 n}}.$$

By easy and well-known arguments, it follows that the function $g_h(x, s) = h(x) \odot s \odot b(x, s)$, where $\odot$ denotes concatenation, is a pseudo-random generator with expansion 1. That is, $g : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$, and for any oracle circuit C of size $2^{(\frac{1}{2}-\frac{5}{3}\gamma_1)n}$, for any "good" permutation h,

$$\left|Prob_{x,s}(C^h(g_h(x, s) = 1)) - Prob_{z\in\{0,1\}^{2n+1}}(C^h(z) = 1)\right| < 2^{-\gamma_2 n}.$$

Step 3: Get double expansion. Based on the function g_h obtained at Step 2, we define $i_h : \{0,1\}^{2n} \to \{0,1\}^{4n}$ by

$$i_h(y) = (s_1, s_2, \ldots , s_{2|y|}),$$

where $s_1, \ldots , s_{2|y|}$ are bits defined inductively as follows: $y_0 = y$ and for $i = 1, \ldots , 2|y|$,

$$\begin{aligned} s_i = & \text{ the first bit of } g_h(y_{i-1}) \text{ and} \\ y_i = & \text{ the last } |y| \text{ bits of } g_h(y_{i-1}). \end{aligned}$$

By an application of the hybrid method, there exists a positive constant γ_3 such that, for any "good" h, and for any oracle circuit C of size $2^{(\frac{1}{2}-2\gamma_1)n}$,

$$\left|Prob_{y\in\{0,1\}^{2n}}(C^h(i_h(y)) = 1) - Prob_{z\in\{0,1\}^{4n}}(C^h(z) = 1)\right| < 2^{-\gamma_3 n}.$$

Step 4: Get exponential expansion. To simplify notation, we fix a "good" permutation h. Let $I_0(y)$ and $I_1(y)$ be the first and respectively the second half of the string $i_h(y)$ defined at Step 3. Let $j = cn$, where c is a constant such that $0 < c < \gamma_3$. Define $F_h : \{0,1\}^{2n} \to \{0,1\}^{2^{cn}}$ as follows. The $\alpha_1\alpha_2 \ldots \alpha_j$ bit of $F_h(y)$ is the first bit of $I_{\alpha_1}(I_{\alpha_2}(\ldots(I_{\alpha_j}(y))\ldots)$. The techniques of Goldreich, Goldwasser, and Micali [GGM86] show that for $\gamma_4 = \gamma_3 - c$, for all good h, for any oracle circuit C of size at most $2^{(\frac{1}{2}-2\gamma_1-c)n}/poly(n)$ (for a fixed $poly$),

$$\left|Prob_{y\in\{0,1\}^{2n}}(C^h(F_h(y)) = 1) - Prob_{z\in\{0,1\}^{2^{cn}}}(C^h(z) = 1)\right| < 2^{-\gamma_4 n}.$$

Observe that for all "good" h, F_h takes an input of size $2n$ and produces an output of size 2^{cn} that cannot be distinguished from a random string except with bias at most $2^{-\gamma_4 n}$ by any oracle circuit of size bounded by 2^{an}, for $a = 1/2 - \eta$ (for an arbitrarily small positive η), working with oracle h. To obtain Theorem 3.3, we only have to choose n such that $m \leq 2^{cn}$, $2^{an} \geq m^{\tau}$ and $2^{-\gamma_4 n} \leq m^{-\beta}$. If τ is sufficiently large, $n = (\tau/a)\log m$ satisfies all these conditions. Let R be the binary string that encodes in the natural way the permutation $h : \{0,1\}^n \to \{0,1\}^n$. We define $f(R, r)$ to be the first m bits of $F_h(r)$. Observe that $|R| = n2^n = poly(m)$ and $|r| = 2n = 2(\tau/a)\log m = 2/(1/2-\eta)\cdot\tau\log m = (4+\alpha)qm$, for an appropriately chosen value of η. It follows that with probability of R at least $1 - 2^{-2^{\Omega(n)}} = 1 - 2^{-poly(m)}$ for any oracle circuit C of size m^τ

$$\left| \frac{1}{2^{2n}} \sum_{r \in \{0,1\}^{2n}} C^R(f(R,r)) - \frac{1}{2^m} \sum_{z \in \{0,1\}^m} C(z) \right| < m^{-\beta}.$$

This concludes the proof of Theorem 3.3. ∎

5 Final comments

Theorem 3.1 opens the possibility of practical implementations of PCP protocols for actual checking of outputs of programs, when we do not suspect the programmer to be malicious, but, maybe, incompetent. In this respect, we note that the "honest" scribe from the proof of Theorem 3.1 needs $O(n^3)$ time for each bit of the proof string (and this can be easily brought down to $O(n^2)$), while the "honest" prover in the full PCP Theorem needs time that is a polynomial of a higher degree (assuming that it has access to a satisfying assignment). Moreover, it is straightforward to program a honest scribe, which is not the case with a honest prover in the full PCP Theorem

We also consider that Theorem 3.1 has some pedagogical value, because (a) it retains some important characteristics of the PCP Theorem, and (b) it has an easier proof. Regarding (a), we note that it retains all the characteristics of the PCP Theorem except one, namely the computational power of the provers: while in the PCP Theorem this power is unlimited, in Theorem 3.1, the provers, given an assignment for a boolean formula, must produce each bit of the proof string in time n^t (t is an arbitrary fixed constant). We remark here that the length of the proof string is $O(n^{(4+\alpha)t})$, and that using an alternative approach in Step 2 in Section 4 based on list decoding of error-correcting codes (as suggested by Sudan [Sud00a]), it can be brought down to $O(n^{(2+\alpha)t})$.

Regarding (b), we list here the main results whose proofs we have omitted: (1) the "easy" part in the proof of the PCP Theorem in which the proof string is exponentially long (Theorem 3.2), (2) Goldreich and Levin construction of one hidden bit for a one-way function (Step 2 in the proof of Theorem 3.3), (3) Yao's hybrid method (Step 3 in the proof of Theorem 3.3), and (4) the construction of Goldreich, Goldwasser, and Micali of a pseudo-random function (Step 4 in the proof of Theorem 3.3). These results are important in their own right and deserve a presentation in a Theory course. Once these results have been demonstrated, the weak variant of the PCP Theorem from Theorem 3.1 follows easily.

One natural question is whether Theorem 3.1 could be used to prove non-approximability results similarly to the PCP Theorem. The answer seems to be negative. However we would like to point out that a variant of the PCP Theorem in which the soundness condition involves polynomial-time provers could be useful. For illustration, let us sketch the arguments by which the PCP Theorem implies that MAX CLIQUE is hard to approximate (unless P = NP). Given a boolean formula ϕ, one can construct in polynomial-time a graph G such that the fraction of the nodes in the maximum clique of G is equal to the maximum acceptance probability of the verifier in a PCP protocol for ϕ (the maximum is taken over all the proof strings). Moreover, from a clique of G having a fraction of α nodes, one can construct a proof string that the verifier accepts with probability at least α. Therefore a good approximation polynomial-time algorithm for MAX CLIQUE can be used to produce in *polynomial-time* a proof string which is accepted by the verifier with probability close to maximum. Since there is a gap between the maximum acceptance probability in the case ϕ is satisfiable and in the case ϕ is not satisfiable, a good approximation algorithm for MAX CLIQUE yields a polynomial-time algorithm for 3SAT. It follows that, in principle, a PCP protocol that is sound against polynomial-time provers can be used to obtain non-approximation results. Theorem 3.1 does not allow such an application because the size of the graph G is larger ($O(n^{(4+\alpha)t}$) than the computational power of the scribes (n^t) (and, thus, if we apply the above argument we only obtain that there is no good approximation algorithm for MAX CLIQUE that runs in time $O(n^{1/(4+\alpha)})$. We formulate as an open problem the discovery of an easy provable version of the PCP Theorem which is sound against polynomial-time provers and which can be used to prove non-approximability results.

6 Acknowledgments

I thank William Gasarch and Lane Hemaspaandra for useful comments. I am grateful to Alina Beygelzimer, Richard Chang, Omer Horvitz, Bala Kalyanasundaram, and Joel Seiferas for helpful discussions.

References

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minumum disclosure proofs of knowledge. *Journal of Computer System Sciences*, 37:156–189, 1988.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct a random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[Gol93] O. Goldreich. Foundations of cryptography (fragments of a book), February 1993. ECCC Technical report, available at http://www.eccc.uni-trier.de/local/ECCC-Books/eccc-books.html.

[Gol99] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudo-randomness.* Springer Verlag, 1999.

[GT00] R. Gennaro and L. Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, 2000.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. Construction of a pseudo-random generator from any one-way function. *SIAM Journal on Computing*, 28(4), 1999.

[Imp96] R. Impagliazzo. Very strong one-way functions and pseudo-random generators exist relative to a random oracle. (manuscript), January 1996.

[Kil92] J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 723–732. ACM Press, 1992.

[Mic94] S. Micali. Cs proofs. In *Proceedings of the 33th IEEE Symposium on Foundations of Computer Science*, pages 436–453, 1994.

[Sga98] J. Sgall. Probabilistic proofs and NP completeness, 1998. Lecture notes available at http://www.math.cas.cz/ sgall/pcp.

[Sud00a] M. Sudan. List decoding: Algorithms and applications (a survey). *Sigact News*, 31(1):16–27, 2000.

[Sud00b] M. Sudan. Probabilistically checkable proofs, July-August 2000. Lecture notes available at http://www.toc.lcs.mit.edu/ madhu/pcp/course.html.

[Tre00] L. Trevisan. Interactive and probabilistic proof-checking. *Annals of Pure and Applied Logic*, 2000. (to appear; available at http://www.cs.berkeley.edu/ luca).

[Zim99] M. Zimand. Sampling under adverse conditions with applications to distributed computing. In *Workshop on Parallel Algorithms, May 1999, Atlanta (satelite workshop of FCRC'99)*, 1999.