8 Probabilistically Checkable Proofs

Consider the problem of checking a proof for a mathematical theorem. Since proofs may be long and difficult to check in their entirety, the question is whether it would be possible to come up with a way of presenting a proof so that only a small part of it has to be checked in order to judge the validity of the theorem with high confidence. This question is addressed by probabilistically checkable proof systems.

Loosely speaking, a probabilistically checkable proof system (PCP) for a language consists of a probabilistic polynomial-time verifier having direct access to individual bits of a binary string. This string (called oracle) represents a proof, and typically will be accessed only partially by the verifier. Queries to the oracle are positions on the bit string and will be determined by the verifier's input and coin tosses (potentially, they might be determined by answers to previous queries as well).

The verifier is supposed to decide whether a given input belongs to the language. If the input belongs to the language, the requirement is that the verifier will always accept given access to an adequate oracle. On the other hand, if the input does not belong to the language, then the verifier will reject with probability at least $1/2$, no matter which oracle is used.

One can view PCP systems in terms of interactive proof systems. That is, one can think of the oracle string as being the prover and of the queries as being the messages sent to him by the verifier. In the PCP setting, however, the prover is considered to be memoryless and thus cannot adjust his answers based on previous queries posed to him.

A more appealing interpretation is to view PCP systems as a possible way of generalizing NP. Instead of conducting a polynomial-time computation upon receiving the entire proof (as in the case of NP), the verifier is allowed to toss coins and query the proof only at locations of his choice. This either allows him to inspect very long proofs (looking at no more than polynomially many locations), or alternatively, look at very few bits of a possible proof.

Most surprisingly, PCP systems have been used to fully characterize the languages in NP. This characterization has been found to be useful in connecting the hardness involved in the approximation of some NP-hard problems with the $\mathrm{P} \neq \mathrm{NP}$ question. In other words, very strong non-appoximability results for various classical optimization problems have been established using PCP systems for NP languages.

8.1 The definition

In the definition of PCP systems we make use of the notion of a probabilistic oracle machine. In our setting, this will be a probabilistic Turing machine which, in addition to the usual features, will have direct access (counted as a single step) to individual bits of a binary string (the oracle).

Definition 8.1 *A* probabilistically checkable proof system for a language L *is a probabilistic polynomial-time oracle machine (called* verifier*), denoted M, satisfying*

- *Completeness: For every $x \in L$ there exists an oracle π_x such that*

$$\Pr[M^{\pi_x} \text{ started with } x \text{ accepts}] = 1$$

- *Soundness: For every $x \notin L$ and every oracle π,*

$$\Pr[M^{\pi} \text{ started with } x \text{ accepts}] \leq \frac{1}{2}$$

where the probability is taken over M's internal coin tosses.

When considering a randomized oracle machine, some complexity measures other than time may come into concern. A natural thing would be to count the number of queries made by the verifier. This number determines what is the portion of the proof being read by the verifier. Another concern would be to count the number of coins tossed by the randomized oracle machine. This in turn determines what is the total number of possible executions of the verifier (once an oracle is fixed).

It turns out that the class of languages captured by PCP systems varies greatly as the above mentioned resources of the verifier are changed. This motivates a quantitative refinement of the definition of PCP systems which captures the above discussed concept.

Definition 8.2 *Let $r, q : \mathbb{N} \to \mathbb{N}$ be integer functions. The complexity class* $\mathrm{PCP}(r(\cdot), q(\cdot))$ *consists of languages having a probabilistically checkable proof system in which it holds that*

- Randomness complexity*: On input $x \in \{0,1\}^*$, the verifier makes at most $r(|x|)$ coin tosses.*

- Query complexity*: On input $x \in \{0,1\}^*$, the verifier makes at most $q(|x|)$ queries.*

For sets of integer functions R and Q, we let

$$\mathrm{PCP}(R, Q) = \bigcup_{r \in R, q \in Q} \mathrm{PCP}(r(\cdot), q(\cdot)) \; .$$

Some remarks before we go on proving results for PCP systems. We will denote by *poly* the set of all integer functions bounded by a polynomial and by *log* the set of all integer functions bounded by a logarithmic function. From now on, whenever referring to a PCP system, we will also specify its corresponding complexity measures.

The definition of PCP involves binary queries to the oracle (which is itself a binary string). These queries specify locations on the string whose binary values are the answers to the corresponding queries. In the following, when given a query q to an oracle π, the corresponding binary answer will be denoted by π_q. Note that an oracle can possibly be of exponential length (since one can specify an exponentially far location on the string using polynomially many bits).

A PCP verifier is called *non-adaptive* if its queries are determined solely based on its input and the outcome of its coin tosses. A general verifier, called *adaptive*, may determine its queries also based on answers to previously received oracle answers. Unless specified otherwise, we will assume the verifier to be adaptive.

8.2 The PCP characterization of NP

The class of languages captured by PCP systems varies greatly as the appropriate parameters $r(\cdot)$ and $q(\cdot)$ are modified. This fact is demonstrated by the following assertions:

- If NP $\subseteq$ PCP$(o(\log), o(\log))$ then NP $=$ P.
- PCP$(poly, poly) =$ NEXP.

By taking either one of the complexity measures to zero the definition of PCP collapses into one of the following degenerate cases:

- PCP$(poly, 0) = co$RP.
- PCP$(0, poly) =$ NP.

When looking at the above degenerate cases of the PCP definition we do not really gain any novel view on the complexity classes involved. Thus, the whole point of introducing the PCP definition may be missed. What we would like to see are more delicate assertions involving both non-zero randomness and query complexity. In the following subsection we demonstrate how PCP systems can be used in order to characterize the complexity class NP in such a non-degenerate way. This characterization will lead to a new perspective on NP and enable us to further investigate the languages in it.

8.3 The PCP Theorem

As already stated, the languages in the class NP are trivially captured by PCP systems using zero randomness and a polynomial number of queries. A natural question arises: can the two complexity measures be traded off, in a way that still captures the class NP? Most surprisingly, not only the answer to the above question is positive, but also a most powerful result emerges. The number of queries made by the verifier can be brought down to a constant while using only a logarithmic number of coin tosses. This result is known as the PCP theorem (it will be cited here without a proof).

Our goal is to characterize NP in terms of PCP systems. We start by demonstrating how NP upper bounds a fairly large class in the PCP hierarchy. This is the class of languages having a PCP system whose verifier makes a polynomial number of queries while using a logarithmic number of coin tosses.

Theorem 8.3 *PCP$(log, poly) \subseteq$ NP.*

Proof. Let L be a language in PCP$(log, poly)$. We will show how to use its PCP system in order to construct a non-deterministic Turing machine M' which decides L in polynomial time. This will imply that L is in NP.

Let M be the probabilistic polynomial time oracle machine in the above PCP$(log, poly)$ system for L. We are guaranteed that on input $x \in \{0,1\}^*$, M makes $poly(|x|)$ queries using $O(\log|x|)$ coin tosses. For the sake of simplicity, we prove the claim here only for a non-adaptive M.

Denote by $\{r_1, \ldots, r_m\}$ the set containing all possible outcomes of the coin tosses made by M (note that $|r_i| = O(\log(|x|))$ and $m = 2^{O(\log(|x|))} = poly(|x|)$). Denote by $(q_1^i, \ldots, q_{n_i}^i)$ the sequence of n_i queries made by M when using the coin sequence r_i (note that n_i is potentially different for each i, and is polynomial in $|x|$). Since M is non-adaptive, its queries are determined as a function of the input x and the coin sequence r_i, and do not depend on answers to previous queries.

By the completeness condition we are guaranteed that for every $x \in L$ there exists a PCP proof π_x such that the verifier M always accepts x when given access to π_x. A natural candidate for an NP-witness for x would be π_x. However, π_x might be of exponential size in $|x|$ and therefore unsuitable to be used as an NP-witness. We will therefore use a compressed version of π_x. This version corresponds to the portion of the proof which is actually being read by the verifier M.

We now turn to the construction of a witness w, given $x \in L$ and a corresponding oracle π_x (for the sake of simplicity we will denote it by π). Consider all possible executions of M on input x given access to the oracle string π (each execution depends on the coin sequence r_i). Take the substring of π containing all the bits $\pi_{q_j^i}$ examined by M during these executions. Encode each entry in this substring as $(index, \pi_{index})$ (that is, $(query, answer)$), and denote the resulting encoded string by w_x^π (note that now $|w_x^\pi|$ is polynomial in $|x|$).

We now describe the non-deterministic Turing machine M' which decides L in polynomial time. Given input x, M' first guesses the part of the oracle needed by M. Let this guess be called w. Afterwards, M' simulates the execution of M on input x for *all* possible r_i's. Every query made by M will be answered by M' according to the corresponding answers appearing in w (if the answer is not in w, then M' guessed incorrectly and rejects). M' will accept if and only if M would have accepted x for *all* possible r_i's.

Since M' simulates the execution of M exactly m times (which is polynomial in $|x|$), and since M is a polynomial time machine, M' itself is a polynomial time machine, as required. It remains to be seen that $L(M')$ indeed equals L:

- For all $x \in L$, we show that there exists a w such that M' started with x accepts. By the perfect completeness condition of the PCP system for L, there exists an oracle π such that $\Pr[M^\pi \text{ accepts}] = 1$. Therefore, it holds that for *all* coin sequences r_i, M accepts x after accessing π. It immediately follows by definition that M' accepts when guessing w_x^π), where w_x^π is as described above.

- For all $x \neq L$, we show that for *all* w's it holds that M' started with x always rejects. By the soundness condition of the PCP system for L, for all oracles π it holds that $\Pr[M^\pi \text{ accepts}] \leq 1/2$. Therefore, for every π, there is at least one coin sequence r_i for which M does not accept x when accessing π. Hence, for every w_x^π guessed by M' there must also be at least one coin sequence r_i for which M' reaches a rejecting state for M, which means that M rejects x.

□

The essence of the above proof is that given a PCP proof (of logarithmic randomness) for some $x \in L$, we can efficiently "pack" it (compress it into polynomial size) and transform it into an NP-witness for x. This is due to the fact that the total portion of the proof used by the verifier (in all possible runs, i.e. over all possible coin sequences) is bounded by a polynomial. Since one can also show that $\mathrm{NP} \subseteq \mathrm{PCP}(log, poly)$ (this will be an assignment), it follows that $\mathrm{NP} = \mathrm{PCP}(log, poly)$. But is this already the end of the story? It turns out that we can actually replace *poly* by $O(1)$! This very surprising result is what we refer to as the PCP theorem.

Theorem 8.4 (PCP Theorem) NP $\subseteq$ PCP$(log, O(1))$.

The PCP theorem is a culmination of a sequence of works, each establishing a meaningful and increasingly stronger statement. The proof of the PCP theorem is one of the most complicated proofs in the theory of computation and it is far beyond our scope to prove it here. We just show here of how one can design verifiers for problems in NP that only need to access a constant number of bits of a proof if a polynomial number of coin tosses is allowed.

Theorem 8.5 NP $\subseteq$ PCP$(poly, O(1))$.

Proof. Besides the SAT problem, it is also known that the 3SAT problem is NP-complete. In the 3SAT problem, we are given a Boolean expression in CNF in which every clause has exactly 3 literals. Such an expression is also said to be in 3-conjunctive normal form, or 3CNF. As an example, consider the expression ϕ with

$$\phi = (x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$$

Hence, if we can show that 3SAT$\in$ PCP$(poly, O(1))$, then also NP $\subseteq$ PCP$(poly, O(1))$.

Consider any expression ϕ in 3CNF. Let $C_1, \ldots, C_n$ be its clauses and $x_1, \ldots, x_m$ be its Boolean variables. Without loss of generality, we assume that $m = n$. Recall that we can arithmetize ϕ using the following transformation rules:

- $\alpha \wedge \beta \rightarrow \alpha \cdot \beta$
- $\neg\alpha \rightarrow (1 - \alpha)$
- $\alpha \vee \beta \rightarrow \alpha * \beta = 1 - (1 - \alpha)(1 - \beta)$

For every clause C_i, let $p_i(x_1, \ldots, x_n)$ be the arithmetized version of the *complement* of C_i over $\mathbb{F}_2$. (For C_1 in ϕ above, $\neg C_1 = \neg x_1 \wedge x_2 \wedge \neg x_4$ and therefore, $p_1(x_1, \ldots, x_n) = (1-x_1)x_2(1-x_4) = x_2 - x_1x_2 - x_2x_4 + x_1x_2x_4 = x_2 + x_1x_2 + x_2x_4 + x_1x_2x_4$ because we are viewing all operations as being in $\mathbb{F}_2$.) Then it holds:

Fact 8.6 *For every $a \in \{0,1\}^n$, a is a satisfying assignment for ϕ if and only if the vector $(p_1(a), \ldots, p_n(a))$ is 0 everywhere.*

We also need another fact:

Fact 8.7 *For any vector $a \in \{0,1\}^n$ with $a \neq 0$ it holds that, for a randomly chosen vector $r \in \{0,1\}^n$,*

$$\Pr[a^T \cdot r \neq 0] = \frac{1}{2}$$

In the following, we assume that all operations are done in $\mathbb{F}_2$. Notice that for any $a, r \in \{0,1\}^n$,

$$\begin{aligned} (p_1(a), \ldots, p_n(a)) \cdot r &= \sum_{i=1}^{n} r_i \cdot p_i(a) \\ &= c(r) + \sum_{i \in S_1(r)} a_i + \sum_{(i,j) \in S_2(r)} a_i a_j + \sum_{(i,j,k) \in S_3(r)} a_i a_j a_k \end{aligned}$$

where $c(r)$, $S_1(r)$, $S_2(r)$, and $S_3(r)$ *only* depend on ϕ and r but *not* on a. Hence, the verifier can compute $c(r)$, $S_1(r)$, $S_2(r)$, and $S_3(r)$ by itself. Let the vectors $s_1(r) \in \{0,1\}^n$, $s_2(r) \in \{0,1\}^{n^2}$, and $s_3(r) \in \{0,1\}^{n^3}$ be defined as

- $(s_1(r))_i = 1$ if and only if $i \in S_1(r)$,
- $(s_2(r))_{i,j} = 1$ if and only if $(i, j) \in S_2(r)$, and
- $(s_3(r))_{i,j,k} = 1$ if and only if $(i, j, k) \in S_3(r)$.

The verifier wants to use these vectors to compute $(p_1(a), \ldots, p_n(a)) \cdot r$ with only a few bit queries of some proof string. For this, it expects a proof string that contains all outcomes for the following three functions, where $a \in \{0,1\}^n$ is a satisfying assignment:

- $A : \{0,1\}^n \to \{0,1\}$ with $A(x) = \sum_{i=1}^n a_i x_i$
- $B : \{0,1\}^{n^2} \to \{0,1\}$ with $B(y) = \sum_{i=1}^n \sum_{j=1}^n a_i a_j y_{i,j}$
- $C : \{0,1\}^{n^3} \to \{0,1\}$ with $C(z) = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n a_i a_j a_k z_{i,j,k}$

Notice that A needs 2^n bits, B needs 2^{n^2} bits, and C needs 2^{n^3} bits to specify all possible outcomes. So the proof string π, which is just a concatenation of the tables of all outcomes for A, B, and C, has a length of $O(2^{n^3})$.

If only proof strings π were allowed that provide correct tables for some linear functions $A(x)$, $B(y)$, and $C(z)$ based on some common $a \in \{0,1\}^n$, then the verifier could simply select a random $r \in \{0,1\}^n$ and ask for $A(s_1(r))$, $B(s_2(r))$, and $C(s_3(r))$. If $c(r) + A(s_1(r)) + B(s_2(r)) + C(s_3(r)) = 0$, the verifier accepts, and otherwise it rejects. If $\phi \in$ 3SAT and $a \in \{0,1\}^n$ is a satisfying assignment for ϕ, the verifier would always accept, and if $\phi \notin$ 3SAT, the fact that there is no satisfying assignment for ϕ and Fact 8.7 imply that the verifier would accept with probability exactly 1/2. Hence, the properties of a PCP could be satisfied if the proof strings were not allowed to have flaws.

However, the given proof string might be flawed, and therefore the verifier has to do some tests before it can be reasonably confident that (at least a large part of) the proof string satisfies the properties above. For this, the verifies executes the following two stages:

1. Verify that A, B, and C in π are what they are supposed to be, namely linear functions defined with respect to the *same* vector $a \in \{0,1\}^n$.
2. Pick an $r \in \{0,1\}^n$ uniformly at random, ask for $A(s_1(r))$, $B(s_2(r))$, and $C(s_3(r))$, and check whether $c(r) + A(s_1(r)) + B(s_2(r)) + C(s_3(r)) = 0$. If so, the verifier accepts, and otherwise it rejects.

The first stage requires a *linearity test*, which checks that the mappings in π are (close to) linear, and a *consistency test*, which checks that the mappings are defined over the same vector a. To perform these tests, we need the following facts.

Fact 8.8 *A function $f : \{0,1\}^n \to \{0,1\}$ is linear if and only if*

$$f(x) + f(x') = f(x + x') \quad \textit{for all } x, x' \in \{0,1\}^n$$

Fact 8.9 *For all $x, x' \in \{0,1\}^{n^2}$, let $y = x \circ x' \in \{0,1\}^{n^2}$ be defined as $y_{i,j} = x_i \cdot x'_j$ for all i, j, and for all $x \in \{0,1\}^n$ and $y \in \{0,1\}^{n^2}$ let $z = x \circ y \in \{0,1\}^{n^3}$ be defined as $z_{i,j,k} = x_i \cdot y_{j,k}$ for all i, j, k. The functions $A(x) = a^T x$ and $B(x) = b^T y$ satisfy*

$$A(x) \cdot A(x') = B(x \circ x') \quad \text{for all } x, x' \in \{0,1\}^n$$

if and only if $b = a \circ a$. Also, $A(x) = a^T x$, $B(x) = b^T y$, and $C(z) = c^T z$ satisfy

$$A(x) \cdot B(y) = C(x \circ y) \quad \text{for all } x \in \{0,1\}^n \text{ and } y \in \{0,1\}^{n^2}$$

if and only if $c = a \circ b$

Fact 8.8 implies the following linearity test:

- pick random $x, x' \in \{0,1\}^n$ and verify that $A(x) + A(x') = A(x + x')$
- pick random $y, y' \in \{0,1\}^{n^2}$ and verify that $B(y) + B(y') = B(y + y')$
- pick random $z, z' \in \{0,1\}^{n^3}$ and verify that $C(z) + C(z') = C(z + z')$

Indeed, it turns out that when using this test a constant number of times, $A(x)$, $B(y)$, and $C(z)$ are close to linear with a reasonably good probability.

Fact 8.9 implies the following consistency test:

- pick random $x, x' \in \{0,1\}^n$ and verify that $A(x) \cdot A(x') = B(x \circ x')$
- pick random $x \in \{0,1\}^n$ and $y \in \{0,1\}^{n^2}$ and verify that $A(x) \cdot B(y) = C(x \circ y)$

Unfortunately, this test is not sufficient because picking x and x' uniformly at random out of $\{0,1\}^n$ does *not* mean that also $x \circ x'$ is picked uniformly at random out of $\{0,1\}^{n^2}$. Fortunately, the linearity condition says that $f(x) = f(x - r) + f(r)$ for any r if f is linear. Hence, replacing each function call $f(x)$ on the right side of the equations in the consistency test by the rule of picking a random r and evaluating $f(x - r) + f(r)$ will achieve the necessary uniformity and therefore make the consistency check effective.

Keeping this trick in mind and inspecting our strategy in stage 2, we discover that there can also be uniformity problems with $s_1(r)$, $s_2(r)$, and $s_3(r)$. To repair that, we replace stage 2 by the following satisfiability test:

- pick a random $r \in \{0,1\}^n$
- pick a random $r_1 \in \{0,1\}^n$ and set $A = A(s_1(r) - r_1) + A(r_1)$
- pick a random $r_2 \in \{0,1\}^{n^2}$ and set $B = B(s_2(r) - r_2) + B(r_2)$
- pick a random $r_3 \in \{0,1\}^{n^3}$ and set $C = B(s_3(r) - r_3) + C(r_3)$
- compute $c(r) + A + B + C$

Now, all the requested table entries are selected uniformly at random. This can be used to show that if the linearity and consistency tests are performed a constant number of times without discovering a flaw, then the probability that the verifier picks flawed table entries when executing the satisfiability test is a sufficiently small constant. Hence, the verifier can be reasonably confident about the outcome of $c(r) + A + B + C$. Repeating the satisfiability test a constant number of times then makes sure that the probability that the verifier accepts an expression ϕ in 3CNF that is not in 3SAT is at most 1/2 so that the PCP conditions are satisfied.

Notice that the verifier never actually has to determine the $a \in \{0,1\}^n$ underlying the functions $A(x)$, $B(y)$, and $C(z)$. It is sufficient for it to know that there *exists* an a so that the functions are (reasonably) consistent with a. More details can be found in a nice survey paper by Hougardy, Prömel, and Steger. $\square$

We state as a side remark that the smallest possible number of queries for which the PCP theorem has been proven is currently 5 (whereas with 3 queries one can get arbitrarily close to soundness error 1/2).

Although it might be tempting to believe that PCP$(log, O(1)) \subseteq$ P, this is not true, because even for a constant number of queries the overall number of positions of an oracle that have to be checked to fulfill the completeness condition can be as high as a polynomial, and therefore it is not feasible to exhaustively enumerate all possible values for the part of the oracle that is used by the verifier.

The conclusion of Theorems 8.3 and 8.4 is that NP is *exactly* the set of languages which have a PCP verifier that asks a constant number of queries using a logarithmic number of coin tosses.

Corollary 8.10 NP = PCP$(log, O(1))$.

Recall that every language L in NP can be associated with an NP-relation R_L. This relation consists of all pairs (x, y) where x is a positive instance of L and y is a corresponding NP-witness. The PCP theorem gives rise to another relation R'_L with some extra properties. In the following we briefly discuss some of the issues regarding the relation R'_L.

Since every $L \in$ NP has a PCP$(log, O(1))$ system, we are guaranteed that for every $x \in L$ there exists a PCP proof π_x such that the corresponding verifier machine M always accepts x when given access to π_x. In order to define our relation, we would like to consider pairs of the form (x, π_x). However, in general, π_x might be of exponential size in $|x|$, and therefore unsuitable to be used in an NP-relation. In order to "compress" it into polynomial size, we can use the construction introduced in the proof of Theorem 8.3. Denote by π'_x the resulting "compressed" version of π_x. We are now ready to define the relation:

$$R'_L = \{(x, \pi'_x) \mid \Pr[M^{\pi_x}(x) = 1] = 1\} \ .$$

By the definition of PCP it is obvious that $x \in L$ if and only if there exists a π'_x such that $(x, \pi'_x) \in R'_L$. It follows from the details in the proof of Theorem 8.3 that R'_L is indeed recognizable in polynomial time.

Although not stated in the theorem, the proof of the PCP theorem actually demonstrates how to efficiently transform an NP-witness y (for an instance x of $L \in$ NP) into an oracle proof $\pi_{x,y}$ for which the PCP verifier always accepts x. Or in other words, there is a polynomial time reduction between the natural NP-relation for L and R'_L.

We conclude that any NP-witness of R_L can be efficiently transformed into an NP-witness of R'_L (i.e. an oracle proof) which offers a trade-off between the portion of the NP-witness being read by the verifier and the amount of certainty it has in its answer. That is, if the verifier is willing to tolerate an error probability of 2^{-k}, it suffices to inspect $O(k)$ bits of the proof. (The verifier just repeats the PCP procedure k times using independent random coin tosses.)

8.4 References

- S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. *Journal of the ACM* 45:501–555, 1998.
- S. Arora, S. Safra. Probabilistically checkable proofs: a new characterization of NP. *Journal of the ACM* 45:70–122, 1998.
- O. Goldreich. Introduction to Complexity Theory. Lecture notes. Dept. of Computer Science and Applied Mathematics, Weizmann Institute.
- O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Algorithms and Combinatorics series (Vol. 17), Springer, 1998.
- S. Hougardy, H.J. Prömel, and A. Steger. Probabilistically checkable proofs and their consequences for approximation algorithms. In *Trends in Discrete Mathematics* (W. Deubner, H.J. Prömel, B. Voigt, eds.), Topics in Discrete Mathematics **9**, North Holland, 1995, pp. 175–223.