# PCP

EKAM KAUR

Abstract. In this paper, we introduce the idea of probabilistically checkable proof systems and their use. We then state the PCP theorem and also its equivalent view of the hardness of approximation.

## 1. Introduction

A probabilistically checkable proof consists of a probabilistic verifier, which for a language $L$ and $x \in L$, can say with certainity that the string is indeed in $L$. For a string not in $L$, it can say this with probability less than $\frac{1}{2}$.

Naively, in a working system, we have both a prover and a verifier. The verifier is an algorithm which first reads the input, and then the proof written by the prover. Then using that information, the verifier outputs whether it accepts or not. Here, the verifier reads the entire proof, and can say with certainty whether the input is in the language.

However, more realistically, we also have randomness. This means that the verifier tosses some coins (in $O(\log n)$ time) in order determine which part of the proof to read. This is done via random access queries.

Now, we cannot say with certainty whether a given input belongs to a language. However, we design it such that the following holds.

**Definition 1.1.** A probabilistically checkable proof system for a language $L$ is given by a verifier $M$ such that both of the following hold:

- Completeness: For every input $x \in L$ there exists a string proof $\pi$, such that the probability that $\pi$ accepts $x$ is 1
- Soundness: For every $x \notin L$ and for every proof $\pi$, the probability that $\pi$ accepts $x$ in less than $\frac{1}{2}$.

Note that the $\frac{1}{2}$ can be improved by repeating the process multiple times.

Given this, it is a natural first question to ask how many queries a PCP system takes to process a string in $L$.

Next, we consider which languages can be modeled by a PCP system. It turns out that this depends largely on what kind of access the system has. Thus, we instead consider the following modified definition below.

**Definition 1.2.** The complexity class $PCP(r(n), q(n))$ consists of all languages such that the following hold:

The verifier processes an input $x$ in at most $r(|x|)$ coin flips, and the verifier makes at most $q(|x|)$ queries to do so.

We can list the following assertions where *poly* is the set of all integer function bounded above by a polynomial and *log* is the set of all functions bounded by a logarithmic function:

*Date*: November 2022.

- **PCP**$(0, 0) =$ **P**
- **PCP**$(O(\log n), 0) =$ **PCP**$(0, O(\log n))) =$ **P**
- **PCP**$(0, \text{poly(n)}) =$ **NP**

Note that since the verifier $M$ runs in polynomial time in $|x|$, the length of $\pi$ should be at most exponential in $|x|$. In fact, we may assume that length of $\pi$ given an input of length $n$, is at most $2^{r(n)}q(n)$.

Also note that here, we say that the verifier is *non-adaptive* (meaning that the set of queries is fixed) rather than adaptive (it can use the result of the previous queries to determine the next index).

## 2. PCP Theorem

We begin by showing the following.

**Lemma 2.1.** ***PCP****(log, poly)* = ***NP***.

*Proof.* Clearly, we have $NP \subseteq \mathbf{PCP}(log, 1)$. For the other direction, let $L \in$ PCP(log, poly) and let $A$ be the verifier for $L$. For given $x \in L$, we will show how to construct a witness for $x$; the $\mathcal{NP}$-machine deciding $L$ will follow naturally. Note that we cannot simply use a "good" proof $\pi_x$ (which is guaranteed to exist since $x \in L$ ) because $\pi_x$ may be exponentially long. However, we can use a "compressed" version of $\pi_x$. In particular, imagine running $A$ for all possible settings of its $O(\log n)$ random coins (here, $n = |x|$). This results in a set $S$ of only polynomially-many indices at which $A$ potentially reads $\pi_x$ (for each setting of its random coins, $A$ reads poly-many indices; there are only $2^{O(\log n)} = \text{poly}(n)$ settings of the random coins). These queries/answers $\{(i, \pi_i)\}_{i \in S}$ will be our $\mathcal{NP}$ witness. Our $\mathcal{NP}$ algorithm for $L$ is simple: on input a witness $w$ of the above form, simulate the computation of $A$ (in the natural way) for all possible settings of its random coins. Accept only if $A$ accepts in all these executions. (If $A$ tries to read an index which is not present in $w$, we count this as a rejection by $A$.) It is not hard to see that this indeed gives a $\mathcal{NP}$ machine deciding $L$.

■

In this section we present the PCP theorem by looking at the satisfiability problem which suffices since that is NP-complete. The main theorem essentially says that every NP language has a PCP verifier that is highly efficient.

First we see what such a system looks like, we give the following examples:

*Example.* The following is "toy" example of the system: First, we define the problem. The gap3SAT problem is the problem of deciding 3CNF formula satisfiability under the promise that the input formula is either satisfiable, or that no assignment satisfies more than half of its clauses.

We construct the following PCP algorithm for it. Given an input formula $\phi$ and proof $\pi$, the PCP verifier samples a uniformly random clause $C$, then reads the assignments for the three variables that appear in $C$ and accepts if and only the assignment satisfies $C$.

We can see that this above example is indeed a valid system. However, it does not use the powerful idea of why PCP systems are useful. What we really want is to determine a global property from making a few local observations. In this case, we want to determine the correctness of the proof while making a few local queries.

We also have the following example:

*Example.* We look at the language GNI consisting of pairs of nonisomorphic graphs in $\mathbf{PCP}(poly, 1)$. Assume that the input is $\langle G_0, G_1\rangle$ both with $n$ nodes. Here the proof $\pi$ is a long array of bits indexed by all possible $n$-vertex graphs where each bit is 0 or 1 depending on which graph it is isomorphic to. We choose arbitrarily if a graph is isomorphic to neither $G_0$ or $G_1$. The verifier picks $G_0$ or $G_1$ and chooses a random permutation of the vertices. It applies this permutation on the graph to obtain an isomorphic graph $H$. Then it gets the corresponding bit of $\pi$ and accepts if and only if it matches the original graph.

If $G_0 \ncong G_1$, then given a proof $\pi$, the verifier accepts with probability 1. If $G_1 \cong G_1$, then it accepts with probability at most $\frac{1}{2}$.

Precisely, the PCP theorem states the following:

**Theorem 2.2.** *(The PCP Theorem)* ***PCP****(log, 1) = NP*

Note that $PCP(r(n), q(n)) \subseteq NTIME(2^{O(r(n))}q(n))$ since a nondeterministic TM can guess the proof in $2^{O(r(n))}q(n)$ time and deterministically verify all the $2^{O(r(n))}$ choices of its random coin tosses. If this verifier accepts for all coin tosses, the nondeterministic machine also accepts. Otherwise it rejects. In particular this implies that $PCP(log, 1) \subseteq NP$ which is the easy direction of the PCP theorem.

Lastly, we also state without proof the following theorem.

**Theorem 2.3.** ***PCP****(poly, poly) =* ***NEXP***

## 3. Hardness of Approximation

Another view of the PCP theorem is related to hardness of approximation. Many combinatorial optimization problems are $NP$ hard to solve exactly, but this view of PCP theorem tells us that it is also hard to even approximate them.

First, we define an the notion of an approximation algorithm.

**Definition 3.1.** (Approximation Algorithm) Take $\rho < 1$. The $\rho$-approximation for $\phi$ is an algorithm $A$ such that for every $3CNF$ formula with $m$ clauses, $A$ gives an assignment satisfying at least $\rho \cdot val(\phi)m$ clauses.

First, consider the following. Given a graph $G$, say we color the vertices with numbers from the set $\{1, \ldots, k\}$. Then we want to find the coloring which maximizes the number of bichromatic edges. Assuming $P \neq NP$, this cannot be done in polynomial time. However, it is also hard to find a coloring which gives a large amount, say $0.999M$, of bichromatic edges (which would be a good substitute for the optimal one). Although, we do not know of such an algorithm, the theory of $NP$-completeness does not rule out the possibility of there existing one.

The PCP theorem, does however, allow us to do the following. We consider a function $f$ mapping an input from an $NP$ complete language to a $k$-graph coloring. This satisfies:

- If the input $x \in L$, then $f(x)$ maps to a satisfying coloring.
- If $x \notin L$, then $f(x)$ is such that no other coloring satisfies 0.999 of the satisfied edges of $f(x)$.

This would then allow us to rule out a 0.999 approximation algorithm.

Next, for concreteness, we use the MAX-3SAT problem to show what the PCP theorem tells us.

**Theorem 3.2.** *(PCP Theorem) There exists some $\rho < 1$ such that for every $L \in NP$, there is some polynomial time function $f$ mapping strings to $3CNF$ formulas such that*

- *(Completeness) If $x$ is in $L$, then val($f(x)$) = 1*
- *(Soundness) If $x$ is not in $L$, then val($f(x)$) $< \rho$.*

*Here val($\phi$) is defined to be the maximum fraction of clauses that can be satisfied via some assignment of variables.*

Note that the existence of a $\rho$ approximation algorithm would imply that $P = NP$. Next, we establish the connection between the two views by first introducing the following lemma

**Lemma 3.3.** *There exists a constant $c$ such that we cannot approximate the number of maximum number of clauses in a 3 CNF formula to a factor of $c$.*

To make this more precise, we give a more formal definition.

**Definition 3.4.** Given a formula $\phi$ and an assignment $\mathbf{a}$ of the variables in $\phi$, let $\mathbf{SAT_a}(\phi)$ denote the fraction of all clauses satisfies by the assignment. Also let **max-SAT**($\phi$) be the maximum value of $S$ if and only if there exists a satisfying assignment.

To see that there is no upper bound, consider the following. Take a unsatisfiable sentence $\phi$ with $m$ clauses. Then it is possible for $\mathbf{max} - \mathbf{SAT}(\phi) = 1 - \frac{1}{m}$, as needed.

To check a lower bound, we claim that $\mathbf{max} - \mathbf{SAT}(\phi) \geq \frac{7}{8}$. To see this, note that given a random assignment $\mathbf{a}$, the probability that it satisfies a given clause is $\frac{7}{8}$. So the expected value of total clauses is satisfies is $\frac{7}{8}$. Thus, there exists an assignment satisfying more than $\frac{7}{8}$ of the clauses.

Note that that a polynomial time algorithm is a $\rho(\cdot)$ approximation algorithm if it always outputs a $\rho(|\phi|)$-approximation for $\phi$.

Also note that a 1 approximation algorithm for the 3-SAT problem would mean we could solve it in polynomial time. We have also already seen that $\frac{7}{8}$ approximation is trivially possible. So how much better can we really do?

To answer this, we first define the following, similar to what we did in the $k$-coloring example:

**Definition 3.5.** (Amplyfying reduction) Let $c < 1$. A $c$-amplifying reduction of 3SAT is a polynomial-time function $f$ on 3CNF formulae such that:

- If $\phi$ is satisfiable, then $f(\phi)$ is satisfiable.
- If $\phi$ is not satisfiable, then every assignment to the variables in $f(\phi)$ satisfies at most a $c$-fraction of the clauses in $f(\phi)$.

Note that the above mapping gives us stability, meaning that small changes in $\phi$ lead to small changes in the solution set.

Relating back to our problem, we have the following lemma showing the hardness of approximation:

**Lemma 3.6.** *Assume $\mathcal{P} \neq \mathcal{NP}$ and that 3 - SAT has a $c^{-1}$-amplifying reduction. Then there is no $c$ approximation algorithm for the 3 - SAT problem.*

*Proof.* We prove this by contradiction. Suppose that there is such an algorithm. Then we can deterministically solve $3SAT$ in polynomial time as follows. On an input $\phi$, run $A(f(\phi))$ to obtain output $k$. If $k \geq c$, output 1, and otherwise output 0. To see that this works, note that if $\phi$ is satisfiable then max-SAT $(f(\phi)) = 1$. So the output $k$ of $A$ must be at least $c$. On the other hand, when $\phi$ is not satisfiable then $\max - \text{SAT}(f(\phi)) < c$ and so the output $k$ of $A$ must satisfy $k < c$. ■

Finally, we establish the connection of this to the PCP theorem with the following theorem showing that they are equivalent.

**Theorem 3.7.** $\mathcal{NP} \subseteq \text{PCP}(\log, 1)$ *iff 3SAT has an amplifying reduction.*

(Note that this implies that it is possible to prove the PCP theorem by constructing an amplifying reduction.)

*Proof.* For the "only if" direction assuming 3SAT has an amplifying reduction, we can construct the following system. First, on an input, the verifier computes $f(\phi)$. To check the proof, the verifier chooses a random clause in $f(\phi)$, query for the assignments to the 3 variables of that clause, and then verify that the clause is satisfied for those settings of the variables. If $\phi$ is satisfiable then $f(\phi)$ is satisfiable which means a valid proof exists. On the other hand, if $\phi$ is not satisfiable then at most a $c$-fraction of the clauses in $f(\phi)$ are satisfiable (for any assignment to the variables), so the verifier accepts with probability at most $c$ regardless of the proof. Since $c$ is a constant, repeating the above procedure a constant number of times (and accepting only if each procedure leads to acceptance) will give the desired soundness error $1/2$ using an overall constant number of queries. Also, the number of random bits needed to select a random clause is logarithmic in $|\phi|$ since $|f(\phi)|$ is polynomial in $|\phi|$.

For the other direction, assume that $NP \subseteq PCP(log, 1)$. Then we describe an amplifying reduction as follows. Let $V$ be the verifier using $c \log n$ random coins and $t$ queries. On an input of a 3CNF formula $\phi$, do the following:

(1) Determine the $t$ indices $q_1, \ldots, q_t$ that the verifier chooses when using random coins $r$.
(2) Run the verifier on all possible settings for the bits of the proof to determine when the verifier accepts. Then we can define a CNF formula $\hat{\phi}_r$ on the variables $x_{q_1}, \ldots, x_{q_t}$ such that $\hat{\phi}_r$ is true exactly when $V$ accepts. Note that the number of clauses in $\hat{\phi}_r$ is constant since $t$ is constant. Then we can convert this using auxiliary variables to a 3CNF formuls $\phi_r$. The number of clauses in this is also constant.

Then we let $f(\phi) = \bigwedge_{r \in \{0,1\}^{c \log n}} \phi_r$. We claim that this is an amplifying reduction. Note that if $\phi$ is satisfiable then $f(\phi)$ is also by completeness of the system. If $\phi$ is not satisfiable, then for any setting of the variables in $f(\phi)$, at least half of all the $\phi_r$ are not satisfied. Let $t'$ denote the maximum number of clauses in any of the $\phi_r$. Then for any setting of the variables, at least $\frac{1}{2}t'$ of the clauses are unsatisfied. This implies that the fraction of satisfied clauses is at most $1 - \frac{1}{2}t'$ which means that $f$ is a $c$-amplifying reduction for any $c > 1 - \frac{1}{2}t'$. ■

## References

[1] Katz, Jonathan *Notes on Complexity Theory* https://www.cs.umd.edu/~jkatz/complexity/f05/lecture12.pdf

[2] Arora, Sanjeev, Barak, Boaz 2009 *Computational Complexity: A Modern Approach* https://theory.cs.princeton.edu/complexity/excerpt.pdf

[3] Scheideler, Christian *Probabilistically Checkable Proofs* John Hopkins University https://www.cs.jhu.edu/~scheideler/courses/600.471_S05/lecture_8.pdf

[4] Song, Min Jae *An introduction to the PCP theorem* 2013. University of Chicago. https://math.uchicago.edu/~may/REU2013/REUPapers/Song.pdf