# Hardness of Approximation, Constructing PCPs

**CS6230: Topics in Information Security**
Lecture 9 (Oct 12, 2021)
Lecture and notes: Prashant Nalini Vasudevan

In the previous lecture, we started looking at Probabilistically Checkable Proofs (PCPs), which are proofs that can be checked by looking at only a small number of bits of the proof. Such proofs are interesting for several reasons – first, they represent a rather surprising answer to the question of what it means to prove something, second, they have found widespread use in proving that optimisation problems are hard to solve even approximately, and third, they are used in cryptography to construct very efficient zero-knowledge proof systems.

One possibility that suggests itself naturally is that PCPs might allow for computationally very efficient verifiers for non-trivial languages, as PCP verifiers only look at a few bits of the proof. However, this is not always the case. Even though the verifier only sees a small part of the proof, often the computation it does to check the proof is comparable to that of an interactive proof or NP proof for the same language. The benefits of a PCP are really the small number of queries and random bits used. Over the next couple of lectures, we will see the aforementioned applications of PCPs that indicate why these are significant.

## 1 The PCP Theorem

Any language that has a PCP where the verifier uses bounded randomness can be decided by a non-deterministic verifier that guesses the PCP proof and checks the probability that the PCP verifier accepts this proof. This implies the following.

**Claim 1.1.** $\mathsf{PCP}[q, \rho] \in \mathsf{NTIME}[q \cdot 2^\rho]$

**Corollary 1.1.** $\mathsf{PCP}[\mathsf{poly}, \log] \in \mathsf{NP}$

The primary question surrounding early research in PCPs was whether a strong converse of Corollary 1.1 was true – whether all languages in $\mathsf{NP}$ had PCPs with small query complexity. It is easy to see that $\mathsf{NP} \subseteq \mathsf{PCP}[\mathsf{poly}, 0]$, as the verifier can just read the entire NP proof, but can we do better than this? In a remarkable sequence of papers in the early 90's, it was shown that we indeed could.

**Theorem 1.2** ([AS92, ALM$^+$92])**.** $\mathsf{NP} \subseteq \mathsf{PCP}[1, \log]$

The number of queries in the PCP in the theorem above can actually be set to 3 (but not 2) [Hås97] (with the soundness error being 0.49 or so). Further, the prover in this PCP for $\mathsf{NP}$ is efficient in the sense that, given a witness, its running time is a polynomial in the length of the PCP proof that it constructs. This is also the case for almost all PCP constructions.

A different, arguably simpler, proof of this theorem was given later by Irit Dinur [Din06]. All of these constructions draw on a number of disparate tools from combinatorics and algebra, involving, among other things, error-correcting codes, expander graphs, random graphs, linearity testing, Fourier analysis, etc.. Covering either of these in any meaningful detail would be beyond the scope of this course, but we will see a simpler construction of much weaker, but still remarkable, PCP for $\mathsf{NP}$.

## 2 Hardness of Approximation

First, though, we will see the application of PCPs to hardness of approximation that spurred this line of research, and which will highlight the importance of constant queries and logarithmic randomness complexity. The simplest example of such an application is to optimising constraint satisfaction problems.

**Definition 2.1.** In the $k$-$\mathsf{CSP}$ problem, an instance consists of a list of $k$-ary constraints over $m$ variables. There are $m$ variables $x_1, \dots, x_m$, and some number $t$ of constraints $\phi_1, \dots, \phi_t$ where each $\phi_i$ is a Boolean formula over $k$ of the $m$ variables – $\phi(x_{i_1}, \dots, x_{i_k})$. The problem, given such an instance $\phi = (\phi_1, \dots, \phi_t)$, is to decide whether there exists an assignment to the variables $x_1, \dots, x_m$ such that all the $\phi_i$'s are satisfied.

This problem is clearly in $\mathsf{NP}$, and is also $\mathsf{NP}$-complete. We will be concerned with the optimisation version of this problem.

**Definition 2.2.** In the $\mathsf{Max}\text{-}k\text{-}\mathsf{CSP}$ problem, given such an instance $\phi = (\phi_1, \dots, \phi_m)$, the task is to find an assignment to $x = (x_1, \dots, x_m)$ that maximises the number of $\phi_i$'s that are satisfied.

Clearly, solving this problem *exactly* would also lead to a solution for $k$-$\mathsf{CSP}$ itself, and so $\mathsf{Max}\text{-}k\text{-}\mathsf{CSP}$ is also $\mathsf{NP}$-hard. But what if we only want to solve it approximately? This is one of the ways to deal with hard problems that show up in practice – to make do with an approximation to the optimal solution. For any $\phi$, let $\mathrm{OPT}(\phi)$ denote the maximum number of constraints in $\phi$ that are satisfied by any assignment. Could there be an efficient algorithm that, given $\phi$, finds an assignment that satisfies more than a $\delta \cdot \mathrm{OPT}(\phi)$ of its constraints? Such an algorithm is said to be a $\delta$-approximation algorithm for $\mathsf{Max}\text{-}k\text{-}\mathsf{CSP}$. PCPs provide a way to show that such even computing such approximation solutions is hard. In particular, the PCP theorem implies the following.

**Theorem 2.1.** *If* $\mathsf{P} \neq \mathsf{NP}$*, for all large enough* $k$*, there is no polynomial-time* $0.51$*-approximation algorithm for* $\mathsf{Max}\text{-}k\text{-}\mathsf{CSP}$*.*

*Proof Sketch.* The idea is to start from an $\mathsf{NP}$-complete problem, and use a $\mathrm{PCP}[1, \log]$ proof for it to reduce it to 0.51-approximating the $\mathsf{Max}\text{-}k\text{-}\mathsf{CSP}$ problem for some $k$. Suppose an $\mathsf{NP}$-complete language $L$ has a PCP proof system with verifier $V$ that makes some constant $k$ queries, and has randomness complexity $c \log n$ on inputs of length $n$. Such a system is guaranteed to exist by the PCP theorem. Fix some input length $n$, and suppose the length of the corresponding PCP proof is $m$.

Given any input $x$ for $L$, and a random string $r \in \{0,1\}^{c \log n}$ of the verifier, consider the algorithm that the verifier runs. It makes queries to $k$ bits of the proof $\pi$, and then computes some function of them that determines whether it accepts or rejects. This specifies a constraint – call it $V_{x,r}(\pi_{r,1}, \dots, \pi_{r,k})$ – on $k$ bits of $\pi$ that determine whether the verifier accepts or rejects. Consider then the set of all of these constraints for all the possible random strings $r$: $V_x = \{V_{x,r}\}_{r \in \{0,1\}^{c \log n}}$. $V_x$ contains at most $2^{c \log n} = n^c$ constraints. By the property of the PCP, this set has the following properties:

- If $x \in L$, then there exists a proof $\pi$ that satisfies all the constraints in $V_x$
- If $x \notin L$, the any possible string $\pi'$ satisfies at most a 0.5-fraction of the constraints in $V_x$

Thus, if there is a polynomial-time algorithm that, given $V_x$ can distinguish between whether there exists a $\pi$ that satisfies either all of the constraints in $V_x$ or at most half of them, then it can be used to decide $L$ in polynomial-time. A 0.51-approximation algorithm for $\mathsf{Max}\text{-}k\text{-}\mathsf{CSP}$ can do this trivially. Thus, if $L$ is indeed not contained in $\mathsf{P}$, then no such polynomial-time algorithm can exist. $\square$

Above, the number of queries being constant allowed the reduction to work for $\mathsf{Max}\text{-}k\text{-}\mathsf{CSP}$ for a constant $k$. This is important because this captures a set of natural and important problems that come up in complexity theory, graph theory, etc.. The $O(\log n)$ bound on the number of random bits allowed the resulting set of constraints to be of polynomial size. If the number of random bits were $\omega(\log n)$, the reduction would no longer work in polynomial time, and we would not be able to use the assumption that $L$ is hard for polynomial-time algorithms.

**Exercise 1.** *Prove that the PCP theorem is, in fact, equivalent to the* $\mathsf{NP}$*-hardness of approximating* $\mathsf{Max}\text{-}k\text{-}\mathsf{CSP}$ *for some constant* $k$*.*

## 3 A Simple PCP

Next, we will see a construction of a simple PCP for $\mathsf{NP}$ that has much worse parameters than those promised by the PCP theorem. Specifically, we will partially prove the following.

**Theorem 3.1.** $\mathsf{NP} \subseteq \mathsf{PCP}[1, \mathsf{poly}]$

While it is indeed much weaker than Theorem 1.2, this is already a remarkable statement that shows that constant-query PCPs can do very non-trivial things. Its proof will illustrate the central components of many PCP constructions, though at a very basic level. We will start by constructing a PCP for a problem that is actually in $\mathsf{P}$ – that of solving a system of linear equations. Later, we will see how this can be extended to a similar PCP for the problem of solving a system of quadratic equations, which is $\mathsf{NP}$-hard.

**Definition 3.1.** In the Multivariate Linear equation problem ($\mathsf{ML}$), an instance consists of a set of $m$ vectors $A = (a_1, \ldots, a_m)$, where each $a_i \in \mathbb{F}_2^n$, and $m$ bits $b = (b_1, \ldots, b_m)$, where $b_i \in \mathbb{F}_2$. The instance $(A, b)$ is in $\mathsf{ML}$ if and only if there exists an $x \in \mathbb{F}_2^n$ such that for all $i \in [m]$, we have $\langle a_i, x\rangle = b_i$.

For simplicity, we will also assume that the matrix $A$ in the instance is always full rank – that is, its rows $a_i$ span the space $\mathbb{F}_2^n$. Clearly $\mathsf{ML} \in \mathsf{NP}$, as the assignment $x$ serves as a witness. Further, $\mathsf{ML} \in \mathsf{P}$, as Gaussian elimination can be used to determine whether such an $x$ exists, and to recover it if it does. So there is a trivial PCP for it where the verifier makes no queries and just solves the problem on its own, but let us ignore that for now.

## 3.1 Encoding a solution

Given a feasible linear system $(A, b)$, suppose it has a solution $u \in \mathbb{F}_2^n$ – that is, $Au = b$. We would like our PCP proof $\pi$ to encode this solution $u$ in a manner that can be verified with few queries. Our encoding will be to simply write down the values of $\langle a, u\rangle$ for all $a \in \mathbb{F}_2^n$. That is, $\pi$ is a string of length $2^n$ that is indexed by $a \in \mathbb{F}_2^n$, and is such that $\pi[a] = \langle a, u\rangle$. Note that this contains, in particular, the evaluations of $\langle a_i, u\rangle$ for all the $a_i$'s in the given system $A$, but also the evaluations of all other possible linear functions. We will denote such an encoding of any string $x \in \mathbb{F}_2^n$ by $H_x$, for reasons that will become clear later.

## 3.2 Verification

Given such a proof string $\pi$ that claims to encode an solution $u$ to the given system $(A, b)$, the task of the PCP verifier can be broken down as follows:

1. Check that $\pi$ is actually equal to $H_u$ for some string $u \in \mathbb{F}_2^n$
2. Check that this encoded string $u$ satisfies $Au = b$

We will defer the first part above for now, and look at how to do the second. Clearly, if $\pi = H_u$, then the statement $Au = b$ can be verified by checking whether, for each $i \in [m]$, $\pi[a_i] = H_u[a_i] = \langle a_i, u\rangle = b_i$. But this requires $m$ queries to the proof, which is bad. Instead, we will use a variation of a technique that we have seen in the past with low-degree polynomials, which is to perform this verification at random points. Corresponding to the fact there that two low-degree polynoimals cannot agree on too many locations, here we have the following.

**Lemma 3.2.** *For any two distinct $u, v \in \mathbb{F}_2^n$,*

$$\Pr_{a \leftarrow \mathbb{F}_2^n}\left[\langle a, u\rangle = \langle a, v\rangle\right] = \frac{1}{2}$$

In other terms, for any distinct $u$ and $v$, the encodings $H_u$ and $H_v$ will agree on exactly half of the $2^n$ coordinates. This is close to a special case of what is known as the Schwarz-Zippel Lemma, which generalises to multivariate polynomials the property that univariate polynomials of degree $d$ can have only $d$ zeroes. To see this, note that the string $H_u$ is actually the truth table of the multivariate linear function $h_u(a) = \langle u, a\rangle$.

**Exercise 2.** *Prove Lemma 3.2.*

We are guaranteed (by the first part of verification, which we skipped) that $\pi$ is an encoding $H_v$ for some vector $v \in \mathbb{F}_2^n$, and wish to check that it is indeed an encoding $H_u$ of a vector $u$ that satisfies $Au = b$. By Lemma 3.2, if we could evaluate $H_u$ at random locations, we could perform this checking by picking a random $a \leftarrow \mathbb{F}_2^n$, and checking that $\pi[a] = H_u[a]$.

And we can indeed compute $H_u[a]$ as, even though we do not know $u$ itself, we know a full rank system of linear equations that it satisfies. So if $a$ is given by some linear combination of the rows of $A$, then $H_u[a] = \langle a, u\rangle$ will simply be the same combination of the entries in $b$. The verifier thus operates as follows:

1. Sample random $r \leftarrow \mathbb{F}_2^m$, and compute $a \leftarrow r^T A$
2. Verify that $\pi[a] = \langle r, b \rangle$

**Claim 3.1.** *If $\pi$ is an encoding $H_v$ of $v \in \mathbb{F}_2^n$ such that $Av \neq b$, then the above procedure rejects with probability* $1/2$.

This probability can then be improved by repetition. Conditioned on $\pi$ actually being an encoding of some string, this claim proves soundness of the PCP. Completeness is easily verified. The verifier here makes only one query, and uses $m$ bits of randomness. It remains, then, to specify how the verifier checks that the proof is indeed such an encoding.

## References

[ALM+92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992* [DBL92], pages 14–23.

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992* [DBL92], pages 2–13.

[DBL92] *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*. IEEE Computer Society, 1992.

[DEL+21] Irit Dinur, Shai Evra, Ron Livne, Alex Lubotzky, and Shahar Mozes. Locally Testable Codes with Constant Rate, Distance, and Locality. `https://simons.berkeley.edu/events/breakthroughs-locally-testable-codes-constant-rate-distance-and-locality`, 2021. [Online; accessed 11-October-2021].

[Din06] Irit Dinur. The PCP theorem by gap amplification. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 241–250. ACM, 2006.

[Hås97] Johan Håstad. Some optimal inapproximability results. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 1–10. ACM, 1997.