# On the Efficiency of Non-Uniform PCPP Verifiers*

Or Meir†

**Abstract**

We define a non-uniform model of PCPs of Proximity, and observe that in this model the non-uniform verifiers can always be made very efficient. Specifically, we show that any non-uniform verifier can be modified to run in time that is roughly polynomial in its randomness and query complexity.

## 1 Introduction

The PCP Theorem [AS98, ALM+98] is one of the major achievements of Complexity Theory. A PCP (Probabilistically Checkable Proof) is a proof that allows checking the validity of a claim by reading only a small number of symbols of the proof. The PCP Theorem asserts the existence of PCPs of polynomial length for any claim that can be stated as membership in an $\mathcal{NP}$ set. The theorem has found many applications, most notably in establishing lower bounds for approximation algorithms.

Further research in the area of PCPs has lead to the more general notion of PCPs of Proximity (PCPPs) [BSGH+06, DR06]. Roghly speaking, in the setting of PCPPs we wish to verify that a given string is close to satisfying some property, by reading only a small part of the string. In order to do so, we are provided a "proof" that is supposed to convince us that the string satisfies the property, but we are only allowed to read a small part of the proof.

One of the interesting aspects of PCPPs is that since the verification procedure reads only a very small part of the tested string, it may run in a time that is smaller than the length of that string. Specifically, it is known that in some cases, a string of length $n$ can be verified in time $\text{poly}(\log n)$ (see, for example, [BFLS91, BSGH+05]).

In this note we consider the case where the verification procedure is non-uniform, and ask how efficient can non-uniform verification procedures be. It turns out that defining a non-uniform model of PCPPs is quite tricky. In particular, if we use the standard non-uniform models of computation, then it is impossible to have efficient verification procedures. We then consider an alternative model, which seems to be more appropriate for this discussion, and show that in this model *every* set that has a PCP of proximity has in fact a very efficient verification procedure.

## 2 Background: PCPs of Proximity

We review a formal definition of PCP of Proximity.

*This research was partially supported by the Israel Science Foundation (grant No. 460/05).

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100 Israel. Email: or.meir@weizmann.ac.il

**Definition 1.** For $x, y \in \{0,1\}^n$, we denote by $\delta(x,y)$ the relative Hamming distance between $x$ and $y$, that is, $\delta(x,y)$ is the fraction of coordinates on which $x$ and $y$ differ. For any string $x \in \{0,1\}^n$ and a set $S \subseteq \{0,1\}^n$ we denote by $\delta(x,S)$ the relative Hamming distance of $x$ to the closest element of $S$. We say that $x$ is $\tau$-far from $S$ if $\delta(x,S) \geq \tau$ and that it is $\tau$-close to $S$ otherwise.

**Notation 2.** Let $M$ be an oracle machine that is given oracle access to two oracles. Then, we denote by $M^{y;z}(x)$ the output of the machine $M$ when given explicit input $x$, oracle access to the oracles $y$ and $z$, where $y$ is the first oracle and $z$ is the second oracle. This notation can be extended to a machine that is given access to three oracles, which will be useful later.

**Definition 3.** Let $S \subseteq \{0,1\}^*$. A probabilistic oracle machine $V$ is said to be a verifier of proximity for $S$ with query complexity $q : \mathbb{N} \to \mathbb{N}$, randomness complexity $r : \mathbb{N} \to \mathbb{N}$, threshold $\tau \in [0,1]$ and rejection probability $\varepsilon \in [0,1]$ if it satisfies the following requirements:

1. $V$ has oracle access to two oracles - the tested oracle and the proof oracle.
2. When given as explicit input a number $n$, the verifier $V$ tosses at most $r(n)$ coins and makes at most $q(n)$ queries (to both its oracles).
3. **Completeness:** For every $x \in S$ there exists a proof string $\Pi_x \in \{0,1\}^*$ such that
$$\Pr\left[V^{x;\Pi_x}(|x|) \text{ accepts}\right] = 1$$
.
4. **Soundness:** For every $x \in \{0,1\}^*$ that is $\tau$-far from $S$ and every proof string $\Pi \in \{0,1\}^*$ it holds that $\Pr\left[V^{x;\Pi}(|x|) \text{ rejects}\right] \geq \varepsilon$.

We say that $V$ has proof length $p : \mathbb{N} \to \mathbb{N}$ if Requirement 3 can be always satisfied with a string $\Pi_x$ such that $|\Pi_x| \leq p(|x|)$.

**Remark 4.** Definition 3 is not the most general that one can consider. Few generalizations that appeared in the PCP literature are

1. PCPPs for pair-languages (see, e.g., [BSGH$^+$06]). In this settings, we let $S \subseteq \{0,1\}^* \times \{0,1\}^*$ and for every $x \in \{0,1\}^*$ we define $S_x = \{y : (x,y) \in S\}$. The verifier gets an explicit input $x \in \{0,1\}^*$ and an oracle access to $y \in \{0,1\}^*$ and to a proof $\Pi \in \{0,1\}^*$, and needs to decide whether $y$ is close to $S_x$.
2. PCPPs with different soundness requirements: One might let $\tau$ and $\varepsilon$ be functions that depend on $|x|$. Also, $\varepsilon$ may depend on $\delta(x,S)$.
3. PCPPs over a different alphabet: One may consider strings over some finite alphabet $\Sigma$ rather than only $\{0,1\}$.

We chose to use Definition 3 for simplicity. However, all the discussion in this note, and in particular Definition 6 and Theorem 7 (below), can be modified to hold for the foregoing generalizations (see Remark 9).

**Remark 5.** Another area for which this discussion is relevant is Property Testing. In the settings of property testing we wish to test that a given string is close to satisfy some property by reading a small part of it, but unlike the case of PCPPs, we need to do so without using a proof oracle. Known examples of property testing include linearity tests (e.g. [BLR93]), low-degree tests (e.g. [RS96]), and graph properties (e.g. [GGR98, GR02]). Although most of the discussion in this note is relevant also to property testing, Theorem 7 (below) holds only in some restricted settings of property testing (see Remark 10).

## 3 Non-uniform verifiers

We now turn our attention to non-uniform verifiers and their efficiency. The use of non-uniform verifiers may be necessary, for example, if the set $S$ is in $\mathcal{P}/\text{poly}$ but not in $\mathcal{P}$. Furthermore, non-uniform verifiers were used in [GS06, Mei07] to verify codes that were constructed using the probabilistic method, and were thus non-uniform. The notion of non-uniform verifiers also arises naturally in some of the transformations presented in [BGS98], but it was avoided there (and reductions to promise problems were used instead).

Defining non-uniform verifiers properly turns out to be a non-trivial issue. The most straightforward way to define such verifiers is to replace the machine $V$ in Definition 3 with a machine that takes advice, or with a family of oracle circuits. However, this definition puts a harsh restriction on the efficiency of the verifiers. For example, if we use machines that take advice, then the running time of those machines will be bounded from below by the length of the advice, because a smaller running time will not allow the machine to read all of its advice. Since in some uses of non-uniform verifiers (e.g. the uses of [GS06, Mei07]) the advice is of length that is polynomial in the length of the tested string, defining verifiers using machines that take advice will not allow the verifier to run in time that is considerably smaller than $|x|$.

In order to avoid this problem, we giving the verifier *oracle access* to the advice string. When using oracle access, the verifier is able to read only the parts of the advice it needs. This means that the running time of the verifier is no longer lower-bounded by the length of the advice string, but rather only in its logarithm. This motivates the following definition.

**Definition 6.** Let $S \subseteq \{0,1\}^*$. A probabilistic oracle machine $V$ is said to be a non-uniform verifier of proximity for $S$ with advice $\{a_n\}_{n\in\mathbb{N}}$ (where $a_n \in \{0,1\}^*$ for all $n \in \mathbb{N}$), query complexity $q : \mathbb{N} \to \mathbb{N}$, randomness complexity $r : \mathbb{N} \to \mathbb{N}$, threshold $\tau \in [0,1]$ and rejection probability $\varepsilon \in [0,1]$ if it satisfies the following requirements:

1. $V$ has oracle access to three oracles: the tested oracle, the proof oracle and the advice oracle.
2. On input $n$, the verifier $V$ tosses at most $r(n)$ coins and makes at most $q(n)$ queries to the tested oracle and the proof oracle. However, $V$ may make an unlimited number of queries to the advice oracle.
3. For every $x \in S$ there exists a proof string $\Pi_x \in \{0,1\}^*$ such that $\Pr\left[V^{x;\Pi_x;a_{|x|}}(|x|) \text{ accepts}\right] = 1$.
4. For every $x \in \{0,1\}^*$ that is $\tau$-far from $S$ and every proof string $\Pi_x \in \{0,1\}^*$ it holds that

$$\Pr\left[V^{x;\Pi_x;a_{|x|}}(|x|) \text{ rejects}\right] \geq \varepsilon$$

We say that $V$ has advice length $\ell : \mathbb{N} \to \mathbb{N}$ if $|a_n| \leq \ell(n)$ for every $n \in \mathbb{N}$. The proof length of $V$ is defined in the same way as in Definition 3.

We conclude this discussion by observing that every set that has a non-uniform verifier, has in fact a very efficient non-uniform verifier, that uses the advice to speed up the verification. This observation, while having an almost trivial proof, makes our lives considerably easier when designing non-uniform verifiers, since it means that we do not have to worry about the running time of the verifiers.

**Theorem 7.** *Let $S \subseteq \{0,1\}^*$ be a set that has non-uniform verifier of proximity $V$ with query complexity $q(n)$, randomness complexity $r(n)$ and proof length $p(n)$. Then $S$ has a non-uniform verifier of proximity $V'$ that runs in time* $\text{poly}\left(q(n), r(n), \log n, \log p(n)\right)$ *and has the same query complexity,*

*randomness complexity, threshold and rejection probability as $V$. Furthermore, the verifier $V'$ uses the same proofs as $V$, and has advice length*

$$2^{r(n)+q(n)} \cdot (q(n) \cdot \log(n + p(n)) + 1)$$

**Remark 8.** Note that the advice length of $V'$ does not depend on the advice length of $V$. Indeed, the construction of $V'$ does not use the advice of $V$ at all.

**Proof** Fix $n \in \mathbb{N}$. The advice string $a_n$ of $V'$ consists of $2^{r(n)+q(n)}$ small strings, each small string corresponding to some pair in $\{0,1\}^{r(n)} \times \{0,1\}^{q(n)}$. Let us denote the small string that corresponds to $(\omega, v) \in \{0,1\}^{r(n)} \times \{0,1\}^{q(n)}$ by $a_n^{(\omega,v)}$. Each small string $a_n^{(\omega,v)}$ is of length $q(n) \cdot \log(n + p(n)) + 1$ and contains the following information:

1. For each $i \in [q(n)]$, the string $a_n^{(\omega,v)}$ contains the $i$-th query that $V$ makes to the tested oracle (or proof oracle) when tossing the coins $\omega$, and when the answers that it got to the first $i-1$ queries are equal to the $i-1$-th prefix of $v$.

2. The string $a_n^{(\omega,v)}$ contains the output of $V$ when tossing coins $\omega$ and when it gets $v$ as the answers to its queries.

Note that indeed $|a_n| \leq 2^{r(n)+q(n)} \cdot (q(n) \cdot \log(n + p(n)) + 1)$ (since $\log(n + p(n))$ is the number of bits required to represent a coordinate in the oracles). Also, observe that the advice of $V'$ does not use the advice of $V$.

We now define the verifier $V'$ in the obvious way: On input $n$, the verifier $V'$ begins by tossing a sequence of coins $\omega \in \{0,1\}^{r(n)}$. Next, for each $i \in [q(n)]$, the verifier $V'$ retrieves the next query that it should make to the tested oracle (or proof oracle) from $a_n$ using $\omega$ and the answers it got for the previous $i-1$ queries. Finally, if the answers that $V'$ got are $v \in \{0,1\}^{q(n)}$, then it retrieves from $a_n^{(\omega,v)}$ the decision of $V$ on coins $\omega$ and on answers $v$, and outputs this decision. It is easy to verify that $V'$ indeed runs in time $\mathrm{poly}(q(n), r(n), \log n, \log p(n))$, and has the required query complexity, randomness complexity, threshold and rejection probability. ■

**Remark 9.** As we mentioned in Remark 4, Theorem 7 can also be made to hold for the various generalizations of PCPPs. However, modifying Theorem 7 to accomodate those generalizations would require suitable changes to its parameters: In particular, generalization to PCPPs for pair-languages would require allowing the verifier to run in time polynomial in the explicit input, and generalization to arbitrary alphabet would require allowing advice length of $2^{r(n)} \cdot |\Sigma|^{q(n)}$.

**Remark 10.** As mentioned in Remark 5, most of the discussion in this note holds also for the area of property testing. However, Theorem 7 does not hold in the standard formalism of property testing. The reason is that Theorem 7 depends crucially on the postulation that the number of queries that the verifier makes is fixed, when given $|x|$. On the other hand, the standard definitions of property testing allow the number of queries to depend on $\delta(x, S)$. However, in a recent paper, Goldreich and Ron [GR08] suggested the model of "Proximity Oblivious Testing" for property testing, in which the number of queries does not depend on $\delta(x, S)$ (while allowing the rejection probability to depend on $\delta(x, S)$). Theorem 7 can be modified to hold in this specific model of property testing.

**Acknowledgement.** The author wishes to thank Oded Goldreich for useful comments on drafts of this note.

## References


[ALM+98] Sanjeev Arora, Carsten Lund, Rajeevi Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and intractability of approximation problems. *Journal of ACM*, 45(3):501–555, 1998. Preliminary version in FOCS 1992.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checkable proofs: A new characterization of np. *Journal of ACM volume*, 45(1):70–122, 1998. Preliminary version in FOCS 1992.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.

[BGS98] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, pcps, and nonapproximability-towards tight results. *SIAM J. Comput.*, 27(3):804–915, 1998.

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[BSGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short pcps verifiable in polylogarithmic time. pages 120–134, 2005.

[BSGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust pcps of proximity, shorter pcps and applications to coding. *SIAM Journal of Computing*, 36(4):120–134, 2006.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards combinatorial proof of the pcp theorem. *SIAM Journal of Computing*, 36(4):155–164, 2006.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[GR02] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[GR08] Oded Goldreich and Dana Ron. On proximity oblivious testing. *Electronic Colloquium on Computational Complexity (ECCC)*, (041), 2008.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and pcps of almost linear length. *Journal of ACM*, 53(4):558–655, 2006. Preliminary version in FOCS 2002, pages 13-22.

[Mei07] Or Meir. Combinatorial construction of locally testable codes. *Electronic Colloquium on Computational Complexity (ECCC)*, (115), 2007.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal of Computing*, 25(2):252–271, 1996.