(poly(log log n), poly(log log n))–Restricted Verifiers are Unlikely to Exist for Languages in $\mathcal{NP}$*

Dimitris Fotakis[1] and Paul Spirakis[1,2]

[1] Computer Engineering and Informatics Dept.
Patras Univ., 265 00 Rion, Patras, Greece. Email: `fotakis@cti.gr`

[2] Computer Technology Institute
Kolokotroni 3, 262 21 Patras, Greece. Email: `spirakis@cti.gr`

Abstract. The aim of this paper is to present a proof of the equivalence of the equalities $\mathcal{NP} = \mathcal{PCP}(\log\log n, 1)$ and $\mathcal{P} = \mathcal{NP}$. The proof is based on producing long pseudo-random bit strings through random walks on expander graphs. This technique also implies that for any language in $\mathcal{NP}$ there exists a restricted verifier using $\log n + c$, c is a constant, random bits. Furthermore, we prove that the equality of classes $\mathcal{NP}$ and $\mathcal{PCP}(\text{poly}(\log\log n), \text{poly}(\log\log n))$ implies the inclusion of $\mathcal{NP}$ in $\mathcal{DTIME}\left(n^{\text{poly}(\log\log n)}\right)$. Also, some technical details of the proof of $\mathcal{NP} = \mathcal{PCP}(\log n, 1)$ are used for showing that a certain class of $(\text{poly}(\log\log n), \text{poly}(\log\log n))$–restricted verifiers does not exist for languages in $\mathcal{NP}$ unless $\mathcal{P} = \mathcal{NP}$.

1 Introduction

A language (decision problem) is in the class $\mathcal{NP}$ if, for every input x which belongs to this language, there exists a membership proof, say π_x, which can be checked in polynomial time by some Turing machine. The majority of typical desicion problems in $\mathcal{NP}$ have concise and simple membership proofs. For example a membership proof for the 3-coloring problem consists of a 3-coloring assignment. Surely, to distinguish a proper 3-coloring from a coloring that is proper on all but one of the vertices, one really has to read the color of every vertex, i.e., the whole proof.

This is not the case with *probabilistically checkable proofs*. Probabilistically checkable proofs are inspected by *verifiers* (polynomial time Turing machines) which proceed as follows: After reading the input x and a random string r, they decide which positions of the proof they want to read. Subsequently, they either accept the input x or reject it — only on the knowledge of (few) queried bits. A language is said to have a probabilistically checkable proof if, for all x in the language, there exists a proof π_x which the verifier accepts for all random strings

* This work was supported by the Information Technology Programme of EU under the project ALCOM–IT.

r, while, for all x not in the language, the verifier rejects all proofs for a majority of the random strings.

Feige et al. [FGLSS91] were the first who used results in the theory of interactive proofs to obtain some non-approximability results for the clique number. Feige et al. exploited a technique of Ajtai, Komlós and Szemerédi [AKS87] and Impagliazzo and Zuckerman [IZ89] for generating long pseudo-random bit strings in order to prove that if $\mathcal{NP} = \mathcal{PCP}(\log n, q(n))$, then the clique number of a graph cannot be approximated within a factor of $2^{\Omega(\log n / q(n))}$ unless $\mathcal{P} = \mathcal{NP}$. Since Arora and Safra [AS92] characterized $\mathcal{NP}$ as $\mathcal{PCP}(\log n, \text{poly}(\log\log n))$, they also proved that it is $\mathcal{NP}$–hard to approximate the clique number within a factor of $2^{\log n / \text{poly}(\log\log n)}$.

Arora et al. [ALMSS92] improved on the work of Arora and Safra by characterized $\mathcal{NP}$ as $\mathcal{PCP}(\log n, 1)$. This result implies [ALMSS92] that MAXSNP–hard problems do not have Polynomial Time Approximation Schemes (PTAS) unless $\mathcal{P} = \mathcal{NP}$, and that for some $\epsilon > 0$ the clique number of a graph cannot be approximated in polynomial time within a factor of n^ϵ unless $\mathcal{P} = \mathcal{NP}$.

Hougardy, Prömel and Steger [HPS94] provide an excellent survey on the recent work concerning probabilistically checkable proofs and their concequences for approximation algorithms.

Relying on [AKS87] and [IZ89] for generating long pseudo-random bit strings by random walks on expander graphs, we further develop the ideas of [AS92] and [ALMSS92] in order to prove that $\mathcal{PCP}(\log\log n, 1) = \mathcal{NP}$ is equivalent to $\mathcal{P} = \mathcal{NP}$. Our techniques also imply that any language in $\mathcal{NP}$ has a restricted verifier that uses exactly $\log n + c$ (c is a constant) random bits and reads $\mathcal{O}(1)$ positions from a membership proof of polynomial length. Furthermore, we prove that $(\text{poly}(\log\log n), \text{poly}(\log\log n))$–restricted verifiers do not exist for languages in $\mathcal{NP}$, unless $\mathcal{NP} \subseteq \mathcal{DTIME}\left(n^{\text{poly}(\log\log n)}\right)$.

The rest of this paper is organized as follows: After some notation and techniques, we prove the fundamental technical lemmas. Then we apply these lemmas to different classes of restricted verifiers. We conclude with some open questions.

1.1 Notation and Techniques

A *verifier* V is a polynomial time Turing machine with access to an input x and a string r of random bits. Furthermore, the verifier has access to a proof π via an oracle, which takes as input a position of the proof the verifier wants to query and outputs the corresponding bit of the proof π.

The result of V's computation, usually denoted by $V(x, r, \pi)$, is either ACCEPT or REJECT. We always assume verifiers to be non-adaptive, that is we assume that the bits a verifier queries solely depend on the random bits r, but not on the outcome of any previously queried bits.

An $(r(n), q(n))$–restricted verifier is a verifier that for inputs x of length n uses at most $\hat{r}(n)$ random bits and inspects at most $\hat{q}(n)$ bits from π, where $\hat{r}(n)$ and $\hat{q}(n)$ are integral functions such that $\hat{r}(n) = \mathcal{O}(r(n))$ and $\hat{q}(n) = \mathcal{O}(q(n))$.

Definition 1 (Arora, Safra [AS92]). A language L is in $\mathcal{PCP}(r(n), q(n))$ iff there exists an $(r(n), q(n))$–restricted verifier V such that:

(a) For all $x \in L$ there exists a proof π_x such that

$$\mathrm{Prob}_r[V(x, r, \pi_x) = \textsc{accept}] = 1 \ ,$$

(b) while for all $x \notin L$ every proof π satisfies

$$\mathrm{Prob}_r[V(x, r, \pi) = \textsc{accept}] < \tfrac{1}{4} \ .$$

□

The probability $\mathrm{Prob}_r[\ldots]$ in Definition 1 is computed with respect to the uniform distribution on $\{0,1\}^{\hat{r}(|x|)}$. Also, the constant 1/4 may be replaced by any constant α between 0 and 1. Let $\mathcal{PCP}_\alpha(.\,,.)$ denote the class of languages defined in the same way as $\mathcal{PCP}(.\,,.)$ except that the constant 1/4 in Definition 1 is replaced by α. Since the probability of getting a wrong answer can be made arbitrarily (but constantly) small by repeating the run of the restricted verifier a (suitable) constant number of times, we have $\mathcal{PCP}_\alpha(.\,,.) = \mathcal{PCP}(.\,,.)$ for any constant α.

A method presented in [AKS87] and [IZ89] for producing long pseudo-random bit strings implies that $\mathcal{NP} = \mathcal{PCP}_{n^{-\epsilon}}(\log n, \log n)$ for some constant $\epsilon > 0$. The following lemma gives the theoretical background for this method:

Lemma 2 (Ajtai, Komlós, Szemerédi [AKS87]). *Let $\mathcal{G}_\varrho$ be an infinite family of d-regular graphs with the following property: If $G = (V, E)$ is a member of $\mathcal{G}_\varrho$ and A denotes its adjacency matrix multiplied by 1/d then all but the largest eigenvalue of A are less than $1 - \varrho$ and positive.*
Then for every subset C of V with $|C| \leq |V|/16$ there exists a constant c such that the probability that a random walk on G of length $\kappa \cdot c$ arrives in every c-th step in a vertex of C is at most $2^{-\kappa}$.

The application of this method is explained in the proof of Lemma 3. The existence of families of graphs $\mathcal{G}_\varrho$ satisfying the requirements of Lemma 2 is based on the existence of *constant degree expanders*. An explicit construction of constant degree expanders is given by Gabber and Galil [GG81]. The so-called Gabber–Galil expander has the advantage that we do not need to explicitly construct the entire graph. In particular, for any vertex in the expander, it is possible to compute the neighbouring vertices in time polynomial in $\log |V|$. Random walks on Gabber–Galil expanders are used (e.g. [MR95]) for obtaining probability amplification results for randomized algorithms.

2 $\mathcal{P} = \mathcal{NP}$ and $\mathcal{NP} = \mathcal{PCP}(\log \log n, 1)$ are Equivalent

Then, we are ready to prove some fundamental technical lemmas.

Lemma 3. *Let L be a language in $\mathcal{PCP}_{1/16}(r(n), q(n))$. Then for* any $0 < \delta < 1$, *L has a restricted verifier R' achieving an error rate δ, using $\hat{r}(n) + f\left(\log \frac{1}{\delta}\right)$ random bits and inspecting $\hat{q}(n) \log \frac{1}{\delta}$ proof bits.*

Proof. Since $L \in \mathcal{PCP}_{1/16}(r(n), q(n))$, there exists a restricted verifier R for L such that achieves an error rate smaller than $1/16$, uses $\hat{r}(n)$ random bits and inspects $\hat{q}(n)$ proof bits. The restricted verifier R' with the desired properties will be the result of multiple invocations of R. Lemma 2 implies that the random bits for multiple invocations of R can be provided by a random walk on a Gabber–Galil expander, under the assumption that $r(n)$ can be bounded from above by a polynomial of n. The restricted verifier R' should proceed as follows:

(1) It should read a random string r of length $\hat{r}(n) + f\left(\log \frac{1}{\delta}\right)$
(2) It should construct [GG81] a regular graph $G(V, E)$ that should be a member of an infinite family of graphs $\mathcal{G}_\varrho$ for which Lemma 2 holds. The graph G should have a vertex for every possible bit string of length $\hat{r}(n)$. Let d be the degree of G and c be the constant of Lemma 2.
(3) R' should perform a random walk (as required by Lemma 2) on the graph G. Let $\kappa \cdot c$ be the length of the random walk. Let r_i be the bit string that is associated with the vertex reached by the random walk at the $(i \cdot c)$-th, $i = 1, \ldots, \kappa$, step. If R' has been invoked as $R'(x, r, \pi)$, then R' should invoke R for each bit string r_i (κ times) as $R(x, r_i, \pi)$. R' should produce ACCEPT iff all invocations of R produce ACCEPT too.

Since error probability of R is less than $1/16$, there exists a $C \subseteq V(G)$, $|C| \leq |V(G)|/16$ such that a vertex $u \in C$ iff it is associated with a bit string r_i that causes invocation $R(x, r_i, \pi)$ to result in wrong outcome. Let κ (the number of times that R is invoked by R') be $\kappa = \log \frac{1}{\delta}$. Then Lemma 2 implies that the probability P_{error} of *all* bit strings r_i causing invocations $R(x, r_i, \pi)$ to result in wrong outcomes will be $P_{error} \leq 2^{-\log(1/\delta)} = 2^{\log \delta} = \delta$.

Furthermore, let $f\left(\log \frac{1}{\delta}\right) = c\left(\log \frac{1}{\delta}\right)(\log d)$. Then a bit string r of length $\hat{r}(n) + f\left(\log \frac{1}{\delta}\right)$ can completely specify the steps of the random walk on G ($\hat{r}(n)$ bits are used for choosing the initial vertex uniformly at random and $c\left(\log \frac{1}{\delta}\right)(\log d)$ bits are used for specifying the $c \log \frac{1}{\delta}$ steps of the walk).

Eventually, since R is invoked by R' $\log \frac{1}{\delta}$ times, the number of proof bits inspected by R' is $\hat{q}(n) \log \frac{1}{\delta}$. □

Remark. Obviously, Lemma 3 can be applied for δ not being constant. So, the characterization of $\mathcal{NP}$ as $\mathcal{PCP}_{n^{-\epsilon}}(\log n, \log n)$ can be obtained [HPS94] by Lemma 3 for $\delta = n^{-\epsilon}$.

Lemma 4. *Let L be any language in $\mathcal{PCP}(r(n), q(n))$. Then there exists a $(r(n), q(n))$–restricted verifier $\hat{R}$ for L that uses exactly $r(n) + t$ random bits, where t does not depend on n.*

Proof. Since $L \in \mathcal{PCP}(r(n), q(n))$, there exists a restricted verifier R achieving error rate smaller than $1/16$, using no more than $\alpha r(n)$, $\alpha > 1$, random bits and inspecting $\hat{q}(n)$ proof bits. Under the assumption that $r(n)$ is bounded from above by a polynomial of n, one can apply Lemma 3 for $\delta = 2^{-(4\alpha+1)}$ in order to obtain a verifier R' that achieves error rate smaller than $\frac{1}{16^\alpha}$, uses $\alpha r(n) + f(4\alpha+1)$ random bits and inspects $(4\alpha+1)\hat{q}(n)$ proof bits. The verifier $\hat{R}$

with the desired properties will be the result of multiple invocations of R'. Under the assumption on the order of $r(n)$, Lemma 2 implies that the random bits for multiple invocations of R' can be provided by a random walk on a Gabber–Galil expander.

Let $\hat{R}(x, r, \pi)$ be an invocation of $\hat{R}$, where r is a random string of length $r(n) + t$. Let κ be the number of times the verifier $\hat{R}$ should invoke the verifier R' and let $t = \kappa \cdot f(4\alpha + 1) + t'$. The verifier $\hat{R}$ interprets the random string r as consisting of the concatenation of the random substrings $r = r^0 \circ r^1 \circ r^2 \circ \cdots \circ r^\kappa \circ r^{\kappa+1}$, where $|r^0| = r(n)$, $|r^i| = f(4\alpha + 1)$, $i = 1, \ldots, \kappa$, and $|r^{\kappa+1}| = t'$. The restricted verifier $\hat{R}$ should proceed as follows:

(1) It should construct [GG81] a regular graph $\hat{G}(\hat{V}, \hat{E})$ that should be a member of an infinite family of graphs $\hat{\mathcal{G}}_\varrho$ for which Lemma 2 holds. The graph $\hat{G}$ should have a vertex for every possible bit string of length $r(n)$. Let $\hat{d}$ be the degree of $\hat{G}$ and $\hat{c}$ be the constant of Lemma 2.

(2) $\hat{R}$ should perform a random walk (as required by Lemma 2) on the graph $\hat{G}$. Let $\alpha \cdot \kappa \cdot \hat{c}$ be the length of the random walk and let m_i be the bit string associated with the vertex reached by the $(i \cdot \hat{c})$-th, $i = 1, \ldots, \alpha \cdot \kappa$, step of the walk. Let r_j be the bit string that consists of the concatenation of
   (a) α consecutive results of the random walk; and
   (b) a substring r^{j+1}, $|r^{j+1}| = f(4\alpha + 1)$, provided by the random string r.
   More precisely $r_j = m_{j\alpha+1} \circ m_{j\alpha+2} \circ \cdots \circ m_{j\alpha+\alpha} \circ r^{j+1}$, $j = 0, \ldots, \kappa - 1$. Additionally, since $|r_j| = \alpha r(n) + f(4\alpha + 1)$, r_j can be used as a random string for the verifier R'.

(3) $\hat{R}$ should invoke $R'(x, r_j, \pi)$, $j = 0, \ldots, \kappa - 1$ (κ times). $\hat{R}$ should produce ACCEPT iff all invocations of R' produce ACCEPT too.

Any bit string r_j consists of the concatenation of two substrings, $r_j = r_j^1 \circ r_j^2$, of length $|r_j^1| = \alpha r(n)$ and $|r_j^2| = f(4\alpha + 1)$. The bit strings r_j^1 are used for choosing the initial vertex of the random walk that the verifier R' performs on the graph G (proof of Lemma 3). By [AKS87] and [MR95], we know that random walks on Gabber–Galil expanders are rapidly mixing. In other words, given any starting vertex, after a small number of steps we expect the random walk to be at a uniformly distributed vertex independent of choice of the initial vertex. The random walk that $\hat{R}$ performs on $\hat{G}$ can be viewed as the composition of $\alpha \cdot \kappa$ different random walks, each generating a different random string m_i. Each of these smaller random walks has constant length $\hat{c}$, whereas we would require $\Theta(\log |\hat{V}|)$ steps to get close to stationary distribution. On the other hand, we choose the initial vertex according to stationary distribution, and this works in our favour. The previous remarks provide the intuition behind the fact that the bit strings r_j^1 are almost uniformly distributed. Further, the bit strings r_j^2 specify the steps of the random walk that the verifier R' performs on graph G. The strings r_j^2 are distributed uniformly at random because they are provided by the random string r. Let $2^{r(n)} = N$ and $2^{\alpha r(n)} = N^\alpha$. Since the error rate of R' is less than $\frac{1}{16^\alpha}$, less than $\left(\frac{N}{16}\right)^\alpha$ bit strings r_j^1 cause the verifier R' to result in wrong outcome.

Let $\hat{C}$ be any set of vertices of $\hat{G}$, $\hat{C} \subseteq \hat{V}(\hat{G})$, such that $|\hat{C}| = |\hat{V}(\hat{G})|/16$. Let w be a random walk of length $\alpha \cdot \hat{c}$ on the graph $\hat{G}$ such that every $\hat{c}$-th step of w reaches only vertices of $\hat{C}$. Let r_w be the bit string of length $\alpha r(n)$ that is produced by the concatenation of the bit strings of length $r(n)$ associated with the vertices (of $\hat{C}$) reached by the $(i \cdot \hat{c})$-th, $i = 1, \ldots, \alpha$, steps of w. The number of different bit strings r_w that can be produced this way is $|\hat{C}|^\alpha = \left(\frac{|\hat{V}(\hat{G})|}{16}\right)^\alpha = \left(\frac{N}{16}\right)^\alpha$. Therefore, since r_j^2 have been specified by the random string r, there exists a $\hat{C} \subseteq \hat{V}(\hat{G}), |\hat{C}| \leq |\hat{V}(\hat{G})|/16$ such that the following proposition holds:

Proposition 5. *There exists an one-to-one correspondence between*

(a) the bit strings r_j^1, $|r_j^1| = \alpha r(n)$, which are the initial substrings of the bit strings r_j, $|r_j| = \alpha r(n) + f(4\alpha + 1)$, that causes the verifier R' to result in wrong outcome; and

(b) the bit strings that can be produced by the concatenation of the bit strings m_i, $i = 1, \ldots, \alpha$, associated with the vertices reached by a random walk w which has length $\alpha \cdot \hat{c}$ and only reaches vertices of $\hat{C}$ at every $\hat{c}$-th step.

Remark. The explicit construction of the set $\hat{C}$ may be computationally intractable because we have no knowledge about the distribution of the "missleading" strings for the verifier R'. The same holds for the set C in the proof of Lemma 3. However, a one-to-one correspondence between the "missleading" strings and the vertices of $\hat{C}$ or C suffices in both cases. So, explicit constructions are required neither in the proof of Lemma 3 nor in the proof of Lemma 4.

Proposition 5 implies that there exists a $\hat{C} \subseteq \hat{V}(\hat{G})$ such that the verifier $\hat{R}$ results in wrong outcome iff the random walk of (2) only reaches vertices of $\hat{C}$. Since Lemma 2 holds for $\hat{G}$ and $|\hat{C}| \leq |\hat{V}(\hat{G})|/16$, the probability that a random walk of length $\alpha \cdot \kappa \cdot \hat{c}$ reaches at every $\hat{c}$-th step a vertex of $\hat{C}$ is at most $2^{-\alpha \cdot \kappa}$. We have already assumed that $\alpha > 1$. Henceforth, the verifier $\hat{R}$ should invoke R' twice ($\kappa = 2$) in order to achieve error rate 1/4.

The complete specification of the random walk requires $r(n)$ random bits for choosing the initial vertex uniformly at random and $2 \cdot \alpha \cdot \hat{c} \cdot \log \hat{d}$ random bits for the specification of the $2 \cdot \alpha \cdot \hat{c}$ steps of the walk. Thus, the verifier $\hat{R}$ requires $r(n) + 2 \cdot f(4\alpha + 1) + 2 \cdot \alpha \cdot \hat{c} \cdot \log \hat{d}$ random bits. If we let $t' = 2 \cdot \alpha \cdot \hat{c} \cdot \log \hat{d}$ and $t = 2 \cdot f(4\alpha + 1) + t'$, then $\hat{R}$ requires exactly $r(n) + t$ random bits, where t does not depend on n. Finally, the bits that $\hat{R}$ should read from the proof π remains $\mathcal{O}(q(n))$ since α is a constant and R' is only invoked twice. □

By applying Lemma 4 for $r(n) = \log n$ and for $r(n) = \log \log n$ we obtain the following corollaries:

Corollary 6. *For any language in $\mathcal{NP}$ there exists a $(\log n, 1)$–restricted verifier that uses exactly $\log n + t$ random bits, where t does not depend on n.*

Corollary 7. *For any language in $\mathcal{PCP}(\log \log n, 1)$ there exists a $(\log \log n, 1)$–restricted verifier that uses exactly $\log \log n + t$ random bits, where t does not depend on n.*

Remark. The proof of Lemma 4 says that random walks of length M in Gabber–Galil expanders with nodes representing $\alpha r(n)$ bits can be emulated by random walks of length αM in Gabber–Galil expanders with nodes representing $r(n)$ bits. Moreover, this holds for α not being constant.

Lemma 8. *Let L be any language over Σ^* and R a $(r(n), q(n))$–restricted verifier for L. Then for any $x \in \Sigma^*$ there exists a SAT instance S_x of size $\mathcal{O}(2^{\hat{r}(n)}\hat{q}(n))$ that is satisfiable iff $x \in L$.*

Proof. For any $x \in \Sigma^*$, $|x| = n$, we will exploit the verifier R in order to construct a SAT instance S_x such that S_x is satisfiable iff x is an element of L.

For every position of a membership proof π we introduce a variable whose values TRUE and FALSE correspond to the values 1 and 0 of the bit at this position. By using these variables the SAT instance S_x is obtained as follows:

- For any possible random string r let S_r denote the boolean formula that expresses which proofs π are accepted by R on input x. Since R queries only $\hat{q}(n)$ bits from the proof, the size of the formulas S_r is $\mathcal{O}(\hat{q}(n))$. Furthermore, since $|r| = \hat{r}(n)$, the number of formulas S_r is $2^{\hat{r}(n)}$.
- Let $S_r^{'}$ be the formula S_r written in conjuctive normal form. Note that each $S_r^{'}$ still contains $\mathcal{O}(\hat{q}(n))$ boolean variables.
- Let S_x be the conjuction of all the formulas $S_r^{'}$. Since formulas $S_r^{'}$ contain $\mathcal{O}(\hat{q}(n))$ variables and the number of $S_r^{'}$ is $2^{\hat{r}(n)}$, the size of the SAT instance S_x is $\mathcal{O}(2^{\hat{r}(n)}\hat{q}(n))$.

□

Since $\mathcal{P} = \mathcal{PCP}(0, 0)$, Lemmas 4 and 8 imply the following:

Theorem 9. $\mathcal{NP} = \mathcal{PCP}(\log\log n, 1) \Leftrightarrow \mathcal{P} = \mathcal{NP}$.

Proof. If $\mathcal{NP} = \mathcal{PCP}(\log\log n, 1)$, then Lemma 4 guarantees that for any language $L \in \mathcal{NP}$ there exists a $(\log\log n, 1)$–restricted verifier R such that $\hat{r}(n) = \log\log n + t$, t is a constant. Also, Lemma 8 implies that the size of the corresponding SAT instance S_x is $\mathcal{O}(\log n)$. Thus, for any $L \in \mathcal{NP}$ and for any $x \in \Sigma^*$ we can decide whether x is in L in polynomial time. □

3 On Verifiers Using poly($\log\log n$) Random Bits

Obviously, Lemma 4 can be applied with α not being constant. Let us consider a $((\log\log n)^\gamma, 1)$–restricted verifier R that uses $\hat{r}(n) = \beta \cdot (\log\log n)^\gamma$ random bits, where β, γ are integral constants, $\beta, \gamma > 1$. Since $r(n)$ can be bounded from above by any polynomial of n, we can apply Lemma 4 to R with parameter $\alpha = \beta \cdot \log\log n$. Consequently, $f(4\alpha + 1) = \mathcal{O}(\log\log n)$. The verifier $\hat{R}$, that is constructed as described by the proof of Lemma 4, uses $\mathcal{O}((\log\log n)^{\gamma-1})$ random bits and invokes the verifier R $\mathcal{O}(\log\log n)$ times. Thus we can apply Lemma 4 to a $((\log\log n)^\gamma, 1)$–restricted verifier in order to obtain a $((\log\log n)^{\gamma-1}, \log\log n)$–restricted verifier. Furthermore, $(\gamma - 1)$ repeated applications of Lemma 4 to a $((\log\log n)^\gamma, 1)$–restricted verifier R result in the construction of a $(\log\log n, (\log\log n)^{\gamma-1})$–restricted verifier $\tilde{R}$. A final application of Lemma 4 to $\tilde{R}$ results in a $(\log\log n, (\log\log n)^{\gamma-1})$–restricted verifier $\hat{R}$ that uses only $\log\log n + t$ random bits, where t does not depend on n. Similarly, γ repeated applications of Lemma 4 to a $((\log\log n)^\gamma, \mathrm{poly}(\log\log n))$–restricted verifier R result in a $(\log\log n, \mathrm{poly}(\log\log n))$–restricted verifier $\hat{R}$ that uses only $\log\log n + t$ random bits.

The previous observations are summarized by the following corollaries:

Corollary 10. *For any language in* $\mathcal{PCP}((\log\log n)^\gamma, 1)$ *where* γ *is an integral constant, there exists a* $(\log\log n, (\log\log n)^{\gamma-1})$*–restricted verifier that uses* $\log\log n + t$ *random bits, where* t *does not depend on* n*.*

Corollary 11. *For any language in* $\mathcal{PCP}(\mathrm{poly}(\log\log n), \mathrm{poly}(\log\log n))$ *there exists a* $(\log\log n, \mathrm{poly}(\log\log n))$*–restricted verifier that uses* $\log\log n{+}t$ *random bits, where* t *does not depend on* n*.*

Corollary 10 implies that under the assumption that $\mathcal{NP}$ is included either in $\mathcal{PCP}((\log\log n)^2, 1)$ or in $\mathcal{PCP}(\log\log n, \log\log n)$, the size of the SAT instance S_x, which is constructed according to the proof of Lemma 8, is $\mathcal{O}(\log n \cdot \log\log n)$. Thus, we have:

Theorem 12. *Unless* $\mathcal{NP} \subseteq \mathcal{DTIME}\left(n^{\log\log n}\right)$*, the class* $\mathcal{NP}$ *is included neither in* $\mathcal{PCP}((\log\log n)^2, 1)$ *nor in* $\mathcal{PCP}(\log\log n, \log\log n)$ *.*

Further, Corollary 11 and the proof of Lemma 8 imply that if the class $\mathcal{NP}$ is included in $\mathcal{PCP}(\mathrm{poly}(\log\log n), \mathrm{poly}(\log\log n))$, then the size of the corresponding SAT instance S_x is $\mathcal{O}(\log n \cdot \mathrm{poly}(\log\log n))$. Thus, we can state the following:

Theorem 13. *Unless* $\mathcal{NP} \subseteq \mathcal{DTIME}\left(n^{\mathrm{poly}(\log\log n)}\right)$*, the class* $\mathcal{NP}$ *is not included in* $\mathcal{PCP}(\mathrm{poly}(\log\log n), \mathrm{poly}(\log\log n))$ *.*

3.1 A Stronger Result for a Certain Class of Verifiers

Then, we use some technical prerequisites in order to prove a slightly stronger negative result about a certain class of $(\log\log n, \mathrm{poly}(\log\log n))$–restricted verifiers for languages in $\mathcal{NP}$.

Let $x, y \in \Sigma^*$ be two bit strings such that $|x| = |y|$. Then x and y are called δ-close iff the fraction of bits on which they differ is less than δ. An *encoding scheme* is a function $E\colon \Sigma^* \mapsto \Sigma^*$ such that for all $x, x' \in \Sigma^*$ with $|x| = |x'|$ the encodings $E(x)$ and $E(x')$ have the same length and coincide in at most $\frac{1}{2}$ of their bits, that is $E(x)$ and $E(x')$ are not $\frac{1}{2}$-close. For any arbitrary but fixed encoding scheme and for any constant $d \in \mathbb{N}$ we can define a new encoding scheme E' with respect to d by

$$E'(y) = (E(y_1), \ldots, E(y_d)) \ , \tag{1}$$

where $y = y_1 \cdots y_d$ and $|y_1| = \ldots = |y_d|$.

A *solution verifier* V is a verifier which in addition has access to a solution s, which it can query via an oracle in the same way as the membership proof π. An $(r(n), q(n), b(n))$*–restricted solution verifier* is a solution verifier which for inputs x of length n uses at most $\hat{r}(n)$ random bits and queries at most $\hat{q}(n)$ blocks of length $\hat{b}(n)$ from s and π, where the starting positions of such blocks are all congruent on modulo $\hat{b}(n)$. Further, $\hat{r}(n)$, $\hat{q}(n)$ and $\hat{b}(n)$ are integral functions such that $\hat{r}(n) = \mathcal{O}(r(n))$, $\hat{q}(n) = \mathcal{O}(q(n))$ and $\hat{b}(n) = \mathcal{O}(b(n))$.

Definition 14 (Arora, Safra [AS92]). Let R be a p-relation and E be an encoding scheme. Then (R, E) is in $\mathcal{PCS}(r(n), q(n), b(n))$ iff there exists an $(r(n), q(n), b(n))$–restricted solution verifier V such that:

(a) For all $x, y \in \Sigma^*$ with $(x, y) \in R$ there exists a proof $\pi_{x,y}$ such that

$$\mathrm{Prob}_r[V(x, r, E(y), \pi_{x,y}) = \text{ACCEPT}] = 1 \ ,$$

(b) For all $x, s \in \Sigma^*$ such that s is not $\frac{1}{4}$-close to the encoding $E(y)$ of a solution $y \in R(x)$ every proof π satisfies

$$\mathrm{Prob}_r[V(x, r, s, \pi) = \text{ACCEPT}] < \tfrac{1}{4} \ .$$

□

The following technical lemma introduces the idea of recursive proof checking. This lemma shows how two solution verifiers can be composed to form a new verifier that queries fewer bits.

Lemma 15 (Arora, Safra [AS92]). *Let $r_i(n)$, $b_i(n)$ be positive functions. If $(R_{3SAT}, \hat{E}) \in \mathcal{PCS}(r_1(n), 1, b_1(n))$ for some encoding scheme $\hat{E}$ and $\tilde{E}$ is an encoding scheme such that for any constant $d \in \mathbb{N}$ the encoding scheme $\tilde{E}'$ given by equation (1) satisfies $(R_{3SAT}, \tilde{E}') \in \mathcal{PCS}(r_2(n), 1, b_2(n))$, then*

$$(R_{3SAT}, \hat{E}) \in \mathcal{PCS}(r_1(n) + r_2(\mathrm{poly}(b_1(n))), 1, b_2(\mathrm{poly}(b_1(n)))) \ .$$

The proof of Lemma 15 is based on techniques first described by Cook in [Coo71]. Let us make the following assumption in order to apply Lemma 15:

Assumption 16. *There exists an encoding scheme E such that*

(a) $(R_{3SAT}, E) \in \mathcal{PCS}(\log\log n, \mathrm{poly}(\log\log n), \mathrm{poly}(\log\log n))$,
(b) there exists a $(\log\log n, \mathrm{poly}(\log\log n), \mathrm{poly}(\log\log n))$*–restricted solution verifier V for (R_{3SAT}, E) which queries the solution string only $\mathcal{O}(1)$ times, and*
(c) if $(R_{3SAT}, E) \in \mathcal{PCS}(\log\log n, 1, \mathrm{poly}(\log\log n))$*, then for any constant $d \in \mathbb{N}$ the encoding scheme E' given by Equation (1) satisfies $(R_{3SAT}, E') \in \mathcal{PCS}(\log\log n, 1, \mathrm{poly}(\log\log n))$.*

Since the solution verifier V uses only $\mathcal{O}(\log\log n)$ random bits, we can assume, without loss of generality, that the length of any proof which can be used by V will be $\mathcal{O}(\mathrm{poly}(\log n))$. Then, we can use V for constructing a new $(\log\log n, 1, \mathrm{poly}(\log\log n))$–restricted solution verifier $\hat{V}$. The idea behind the construction of $\hat{V}$ is simple: Using the proof π that V would read, we construct $\hat{V}$ that uses a proof $\hat{\pi}$ which contains for every possible random string r the sequence of $\mathrm{poly}(\log\log n)$ bits that V would read from π on input x and random string r. For the verification process the verifier $\hat{V}$ reads a consecutive sequence of $\mathrm{poly}(\log\log n)$ bits from $\hat{\pi}$ that depend on r and uses these bits to determine what the verifier V would have answered if it had received these bits as answers for its queries to π.

The new proof $\hat{\pi}$ for $\hat{V}$ contains the encoding of π under the E_1 encoding scheme that is defined as follows:

Definition 17 (Babai et al. [BFLS91]). Let x be a bit vector of length l, $x \in \mathbb{F}_2^l$. Let p be the smallest prime larger than $\log^2 l$. We may assume that $\log l$ and $\log\log l$ are integer (otherwise elongate x by a suitable number of bits). Set $m = \lceil \log l / \log\log l \rceil$. Let H be a subset of $\mathbb{F}_p$ of size $\log l$. Note that $|\mathrm{H}|^m \geq l$. Thus, we may interpret x as a function $f_x : \mathrm{H}^m \mapsto \{0,1\}$. Then $E_1(x) : \mathbb{F}_p^m \mapsto \mathbb{F}_p$ is the low degree extension of f_x. $E_1(x)$ is a polynomial of degree at most $|\mathrm{H}|$ in each variable and agrees with f_x on H^m. □

The use of the encoding scheme E_1 provides a restricted solution verifier with efficient polynomial testers like Low Degree Test [RS92] and Extended LFKN-Test [LFKN92]. The construction of the verifier $\hat{V}$ from V is similar to the construction found in the proof of Theorem 18 ([ALMSS92, HPS94]). Theorem 18 is used in the proof of $\mathcal{NP} = \mathcal{PCP}(\log n, 1)$ and exploits the encoding scheme E_1 and the Low Degree Test.

Theorem 18 (Arora et al. [ALMSS92]).
If a $(\log n, \mathrm{poly}(\log n), \mathrm{poly}(\log n))$–restricted solution verifier that queries the solution string a constant number of times exists for (R_{3SAT}, E_1), then $(R_{3SAT}, E_1) \in \mathcal{PCS}(\log n, 1, \mathrm{poly}(\log n))$.

The verifier $\hat{V}$ implies that $(R_{3SAT}, E) \in \mathcal{PCS}(\log\log n, 1, \mathrm{poly}(\log\log n))$. Furthermore, we have assumed the encoding scheme E to satisfy the requirements of Lemma 15. So, we can apply Lemma 15 with respect to the encoding scheme $\hat{E} = \tilde{E} = E$ and the functions $r_1(n) = r_2(n) = \log\log n$ and $b_1(n) = b_2(n) = \mathrm{poly}(\log\log n)$. We therefore deduce that $(R_{3SAT}, E) \in \mathcal{PCS}(\log\log n, 1, \mathrm{poly}(\log\log\log\log n))$.

The following lemmas are also proved in [ALMSS92]:

Lemma 19 (Arora et al. [ALMSS92]). *Let E_0 denote the encoding scheme given by $E_0 : y \mapsto \{y^T z\}_{z \in \mathbb{F}_2^{|y|}}$. Then $(R_{3SAT}, E_0) \in \mathcal{PCS}(n^3, 1, 1)$.*

Lemma 20 (Arora et al. [ALMSS92]). *Let E_0' be the encoding scheme defined by Equation (1). Then for all constants $d \in \mathbb{N}$ one has $(R_{3SAT}, E_0') \in \mathcal{PCS}(n^3, 1, 1)$.*

Lemmas 19 and 20 imply that we can apply Lemma 15 with respect to the encoding schemes $\hat{E} = E$ and $\tilde{E} = E_0$ and to the functions $r_1(n) = \log\log n$, $r_2(n) = n^3$ and $b_1(n) = \text{poly}(\log\log\log\log n)$, $b_2(n) = 1$. We therefore deduce that $(R_{3SAT}, E) \in \mathcal{PCS}(\log\log n, 1, 1)$. Obviously, this implies that 3SAT is in $\mathcal{PCP}(\log\log n, 1)$. Since 3SAT is an $\mathcal{NP}$*–complete* language [GJ79], we have $\mathcal{NP} = \mathcal{PCP}(\log\log n, 1)$ and subsequently $\mathcal{P} = \mathcal{NP}$. So, we have proved that:

Theorem 21. *If there exists an encoding scheme E that satisfies Assumption 16, then $\mathcal{P} = \mathcal{NP}$.*

Under Assumption 16, Corollary 11 implies that Theorem 18 can also be applied to a $(\text{poly}(\log\log n), \text{poly}(\log\log n), \text{poly}(\log\log n))$–restricted solution verifier that uses membership proofs of length $\mathcal{O}(\text{poly}(\log n))$.

Moreover, Assumption 16.b can be removed if we handle the solution string s (that is of length $\mathcal{O}(\text{poly}(\log n)))$ as the proof π in the construction of the verifier $\hat{V}$. Furthermore, it is plausible to assume that a properly encoded solution can be distinguished in any proof used by a restricted verifier for a language in $\mathcal{NP}$. The combination of Theorem 18 with the previous observations provides strong evidence that unless $\mathcal{P} = \mathcal{NP}$, the class $\mathcal{NP}$ is not included in $\mathcal{PCP}(\log\log n, \text{poly}(\log\log n))$. The same holds for the classes $\mathcal{NP}$ and $\mathcal{PCP}(\text{poly}(\log\log n), \text{poly}(\log\log n))$ if there exists a $(\text{poly}(\log\log n), \text{poly}(\log\log n))$–restricted verifier that uses membership proofs of length $\mathcal{O}(\text{poly}(\log n))$.

4 Conclusions – Open Problems

Corollaries 10 and 11 combined with the proof of Lemma 8 are used for showing that unless the class $\mathcal{NP}$ is contained in $\mathcal{DTIME}\left(n^{\text{poly}(\log\log n)}\right)$, $\mathcal{NP}$ is not included in $\mathcal{PCP}(\text{poly}(\log\log n), \text{poly}(\log\log n))$. The question that naturaly arises is whether the equality of classes $\mathcal{P}$ and $\mathcal{NP}$ is implied by the existence of $(\text{poly}(\log\log n), \text{poly}(\log\log n))$–restricted verifiers for languages in $\mathcal{NP}$. We prove that the answer is positive for restricted solution verifiers using membership proofs of length $\mathcal{O}(\text{poly}(\log n))$ and satisfying Assumptions 16.b and 16.c. These assumptions are not of major importance and may be removed. Concerning languages in the class $\mathcal{NP}$, restricted solution verifiers are not expected to be more powerful than ordinary restricted verifiers. However, there is no proof about the equivalence of the equalities $\mathcal{NP} = \mathcal{PCP}(\text{poly}(\log\log n), \text{poly}(\log\log n))$ and $\mathcal{P} = \mathcal{NP}$.

The existence of a restricted verifier that uses membership proofs of length $\mathcal{O}(\text{poly}(\log n))$ for a language in $\mathcal{NP}$ implies the inclusion of $\mathcal{NP}$ in $\mathcal{DTIME}\left(n^{\text{poly}(\log n)}\right)$. Another open question is whether the existence of such a restricted verifier implies the inclusion of $\mathcal{NP}$ in $\mathcal{P}$.

Lemma 4 can be applied to a $(\log n, 1)$–restricted verifier in order to construct a $\left(\frac{\log n}{\log\log n}, \log\log n\right)$–restricted verifier for any language in $\mathcal{NP}$. However, the existence of $\left(\frac{\log n}{\log\log n}, 1\right)$ and $\left(\frac{\log n}{\text{poly}(\log\log n)}, 1\right)$–restricted verifiers for languages in $\mathcal{NP}$ are open problems.

References

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in logspace. *Proc. of the 19th ACM Symposium on Theory of Computing*, pp. 132–140, 1987.

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Proc. of the 33th Annual IEEE Symposium on Foundations of Computer Science*, pp. 14–23, 1992.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of $\mathcal{NP}$. *Proc. of the 33th Annual IEEE Symposium on Foundations of Computer Science*, pp. 2–13, 1992.

[BFLS91] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. *Proc. of the 23th ACM Symposium on Theory of Computing*, pp. 21–31, 1991.

[Coo71] S.A. Cook. The complexity of theorem-proving procedures. *Proc. of the 3rd ACM Symposium on Theory of Computing*, pp. 151–158, 1971.

[FGLSS91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost $\mathcal{NP}$–complete. *Proc. of the 32th Annual IEEE Symposium on Foundations of Computer Science*, pp. 2–12, 1991.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear–sized superconcentrators. *Journal of Computer and System Sciences* **22**, pp. 407–420, 1981.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, 1979.

[HPS94] S. Hougardy, H.J. Proömel, and A. Steger. Probabilistically Checkable Proofs and their Consequenses for Approximation Algorithms. *Discrete Mathematics* **136**, pp. 175–223, 1994.

[IZ89] T. Impagliazzo and D. Zuckerman. How to recycle random bits. *Proc. of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pp. 248–253, 1989.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the Assosiation for Computing Machinery* **39** (4), pp. 859–868, 1992.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, 1995.

[RS92] R. Rubinfeld and M. Sudan. Testing Polynomial Functions Efficiently and over Rational Domains. *Proc. 3rd Annual ACM–SIAM Symposium on Discrete Algorithms*, pp. 23–32, 1992.