This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

4. Proof Checking and Non-Approximability

Stefan Hougardy

4.1 Introduction

In this chapter we will present the PCP-Theorem and show how it can be used to prove that there exists no PTAS for $\mathcal{APX}$-complete problems unless $\mathcal{P} = \mathcal{NP}$. Moreover we will show how the PCP-Theorem implies that MaxClique cannot be approximated up to a factor of n^{ε} in an n-vertex graph and how this factor can be improved to $n^{1-\varepsilon}$ by making use of Håstad's [Hås97a] result showing that δ amortized free bits suffice for a $(\log n, 1)$-verifier to recognize any $\mathcal{NP}$ language.

4.2 Probabilistically Checkable Proofs

The class PCP (standing for **P**robabilistically **C**heckable **P**roofs) is defined as a common generalization of the two classes co-$\mathcal{RP}$ and $\mathcal{NP}$. To see this let us briefly recall the definitions of these two classes as given in Chapter 1. A language L belongs to co-$\mathcal{RP}$ if there exists a randomized polynomial time Turing machine M such that

$x \in L \quad \Rightarrow \text{Prob}[M \text{ accepts } x] = 1.$

$x \notin L \quad \Rightarrow \text{Prob}[M \text{ accepts } x] < 1/2.$

Using a similar notation, the class $\mathcal{NP}$ can be defined to consist of all languages L for which there exists a polynomial time Turing machine M such that

$x \in L \quad \Rightarrow \exists$ certificate c such that M accepts (x, c).

$x \notin L \quad \Rightarrow \forall$ certificates c M rejects (x, c).

It is easily seen that this definition is equivalent to the one given in Chapter 1 as the certificate c used in the above definition simply corresponds to an accepting computation path of the non-deterministic Turing machine used in Chapter 1 to define the class $\mathcal{NP}$.

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

The idea of the class PCP now is to allow simultaneously the use of randomness and non-determinism (or equivalently use of a certificate). The notions here are slightly different: the Turing machine will be called *verifier* and the certificates are called *proofs*.

A *verifier* V is a (probabilistic) polynomial time Turing machine with access to an input x and a string τ of random bits. Furthermore the verifier has access to a proof π via an oracle, which takes as input the position of the proof the verifier wants to query and outputs the corresponding bit of the proof π. Depending on the input x, the random string τ and the proof π the verifier V will either accept or reject the input x. We require that the verifier is *non-adaptive*, i.e., it first reads the input x and the random bits τ, and then decides which positions in the proof π it wants to query. Especially this means that the positions it queries do not depend on the answers the verifier got from previous queries. We will denote the result of V's computation on x, τ and π as $V(x, \tau, \pi)$.

As we will see later, verifiers are very powerful if we allow them to use polynomially many random bits and to make polynomially many queries to a proof. This is the reason why we introduce two parameters that restrict the amount of randomness and queries allowed to the verifier. An $(r(n), q(n))$-*restricted verifier* is a verifier that for inputs of length n uses at most $\hat{r}(n) = \mathcal{O}(r(n))$ random bits and queries at most $\hat{q}(n) = \mathcal{O}(q(n))$ bits from the proof π.

The class $\mathrm{PCP}(r(n), q(n))$ consists of all languages L for which there exists an $(r(n), q(n))$-restricted verifier V such that

$$x \in L \quad \Rightarrow \quad \exists \pi \text{ s. t. } \mathrm{Prob}_\tau[V(x, \tau, \pi) = ACCEPT] = 1. \qquad (\textit{completeness})$$

$$x \notin L \quad \Rightarrow \quad \forall \pi \ \mathrm{Prob}_\tau[V(x, \tau, \pi) = ACCEPT] < 1/2. \qquad (\textit{soundness})$$

Here the notation $\mathrm{Prob}_\tau[\ldots]$ means that the probability is taken over all random strings τ the verifier may read, i.e., over all 0-1-strings of length $\hat{r}(n) = \mathcal{O}(r(n))$.

We extend the definition of PCP to sets R and Q of functions in the natural way: $\mathrm{PCP}(R, Q) = \cup_{r \in R, q \in Q} \mathrm{PCP}(r(\cdot), q(\cdot))$.

Obviously, the class PCP is a common generalization of the classes co-$\mathcal{RP}$ and $\mathcal{NP}$, since we have:

$$\mathrm{PCP}(\mathrm{poly}, 0) = \text{co-}\mathcal{RP}$$

$$\mathrm{PCP}(0, \mathrm{poly}) = \mathcal{NP}$$

Here we denote by poly the set of all polynomials; similarly we use polylog for the set of all poly-logarithmic functions etc.

The natural question now is: How powerful is a (poly, poly)-restricted verifier ? As was shown by Babai, Fortnow and Lund in 1990 [BFL91] this verifier is *very* powerful; namely they proved that

$$\mathrm{PCP}(\mathrm{poly}, \mathrm{poly}) = \mathcal{NEXP}$$

This result was "scaled down" almost at the same time independently by Babai, Fortnow, Levin and Szegedy [BFLS91] who proved

$$\mathcal{NP} \subseteq \mathrm{PCP}(\mathrm{poly}\log n, \mathrm{poly}\log n)$$

and by Feige, Goldwasser, Lovász, Safra and Szegedy [FGL+91] who proved that

$$\mathcal{NP} \subseteq \mathrm{PCP}(\log n \cdot \log\log n, \log n \cdot \log\log n).$$

The hunt for the smallest parameters for the class PCP that were still able to capture all of $\mathcal{NP}$ was opened. A natural conjecture was that $\mathcal{NP}$ equals $\mathrm{PCP}(\log n, \log n)$ and indeed, shortly after Arora and Safra [AS92] proved an even stronger result that broke the $\log n$ barrier in the number of query bits:

$$\mathcal{NP} = \mathrm{PCP}(\log n, \mathrm{poly}\log\log n).$$

Just some weeks later the hunt was brought to an end by Arora, Lund, Motwani, Sudan and Szegedy [ALM+92] who obtained the ultimate answer in the number of queries needed to capture $\mathcal{NP}$:

Theorem 4.1 (The PCP-Theorem). $\mathcal{NP} = \mathrm{PCP}(\log n, 1)$

At first sight this is a very surprising result as the number of queries needed by the verifier is a *constant*, independent of the input size. Let us take as an example the problem 3SAT and consider the usual way to prove that a 3SAT instance x is satisfiable, namely a truth assignment to the variables. If we are allowed to query only a constant number of the values assigned to the variables, then it is impossible to decide with constant error probability whether the given 3SAT instance is satisfiable. The reason why this does not contradict the PCP-Theorem is the following. The proofs used by the verifiers are not the kind of certificates that are usually used for problems in $\mathcal{NP}$. Rather as proofs there will be used special error correcting encodings of such certificates.

The proof of the PCP-Theorem will be given in Chapter 5. The main part in the proof is to show that the inclusion $\mathcal{NP} \subseteq \mathrm{PCP}(\log n, 1)$ holds. The other inclusion can easily be derived from the following slightly more general statement.

Proposition 4.2. $\mathrm{PCP}(r(n), q(n)) \subseteq \mathrm{NTIME}(2^{\mathcal{O}(r(n))} \cdot \mathrm{poly})$ *whenever* $q(n) = \mathcal{O}(\mathrm{poly})$.

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

Proof. Simply observe that we can guess the answers to the $\mathcal{O}(q(n))$ queries to the proof π and simulate the PCP-verifier for all $2^{\mathcal{O}(r(n))}$ possible random strings to decide whether we should accept an input x or not. Each of the simulations can be carried out in polynomial time. ■

Interestingly, there can be proved a trade off between the randomness and the number of queries needed by the PCP-verifier such that its power is always exactly suited to capture the class $\mathcal{NP}$[Gol95].

Proposition 4.3. *There exist constants $\alpha, \beta > 0$ such that for every integer function $l(\cdot)$, so that $0 \leqslant l(n) \leqslant \alpha \log n$*

$$\mathcal{NP} = \mathrm{PCP}(r(\cdot), q(\cdot))$$

where $r(n) = \alpha \log n - l(n)$ and $q(n) = \beta 2^{l(n)}$.

As the extreme cases this proposition shows $\mathcal{NP} = \mathrm{PCP}(\log n, 1)$ and $\mathcal{NP} = \mathrm{PCP}(0, \mathrm{poly})$.

4.3 PCP and Non-Approximability

The reason why the PCP-Theorem caused a sensation is due to its close connection to the seemingly unrelated area of approximation algorithms. This connection was first discovered by Feige, Goldwasser, Lovász, Safra and Szegedy [FGL+91] who observed that if the problem MaxClique can be approximated up to a constant factor then $\mathrm{PCP}(r(n), q(n)) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(r(n)+q(n))} \cdot \mathrm{poly})$. This observation was the main motivation for proving $\mathcal{NP} \subseteq \mathrm{PCP}(\log n, \log n)$ and thus showing that no constant factor approximation algorithm for MaxClique can exist, unless $\mathcal{P}=\mathcal{NP}$.

Later on, the connection between PCPs and non-approximability has been extended to a large number of other optimization problems. The results obtained in this area are always of the type: No "good" polynomial time approximation algorithm can exist for solving the optimization problem, unless something very unlikely happens. Here the term "very unlikely" means that the existence of such an approximation algorithm would imply $\mathcal{P}=\mathcal{NP}$ or $\mathcal{NP}=\mathcal{ZPP}$ or $\mathcal{NP} \subseteq \mathrm{DTIME}(2^{\mathrm{poly}\log n})$ or similar statements. The precise definition of a "good" approximation algorithm heavily depends on the underlying optimization problem. Roughly, optimization problems can be divided into three classes: (1) Problems for which polynomial time constant factor approximation algorithms are known, but no PTAS can exist. As an example we will show in Section 4.4 that Max3Sat has no PTAS and therefore no problem in $\mathcal{APX}$ has a PTAS unless $\mathcal{P}=\mathcal{NP}$. (2) Problems that can be approximated up to a factor of $c_1 \cdot \log n$ but no polynomial time algorithm can achieve an approximation ratio of

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

$c_2 \cdot \log n$, for certain constants $c_1 > c_2$. In Chapter 10 we will see that the problem SETCOVER is an example for such an optimization problem. (3) Problems whose solution is of size $\mathcal{O}(n)$ but that cannot be approximated up to a factor of n^ε for certain constants ε. Famous examples in this class are MAXCLIQUE and CHROMATICNUMBER. We will prove in Section 4.6 that no polynomial time approximation algorithm for MAXCLIQUE can have a performance guarantee of n^ε for a certain ε, unless $\mathcal{P}=\mathcal{NP}$.

For more than twenty years no non-trivial lower bounds for the approximation guarantee of any of the above mentioned problems was known. The PCP-Theorem now gave such results for a whole bunch of optimization problems. But this is not the only consequence set off by the PCP result. Soon after getting the first non-trivial non-approximability results people started to tighten the gap between the best known approximation factors achievable in polynomial time and the lower bounds obtained by using the PCP-Theorem. Surprisingly, this challenge not only led to improvements on the just obtained new lower bounds, but also for several classical optimization problems, better polynomial time approximation algorithms have been found. As an example Goemans and Williamson [GW94a] improved the long known trivial 2-approximation algorithm for MAXCUT to an 1.139-approximation algorithm for this problem. Interestingly, the new approximation algorithms do not rely on the PCP result. The PCP-Theorem and its consequences for non-approximability results rather functioned as a new strong motivation to try to improve the best known approximation algorithms known so far.

For improving the lower bounds for non-approximability results by using the PCP-Theorem it turns out that one has to reduce the constants involved in the $\mathcal{O}$-terms of the number of random bits and query bits the verifier makes use of, while at the same time the probability of accepting wrong inputs has to be lowered.

In the definition of the class PCP we required the verifier to accept wrong inputs with error probability of at most $1/2$. However, this number $1/2$ is chosen arbitrarily: by repeating the verification process a constant number of times, say k times, the error probability can easily be decreased to $(1/2)^k$ while changing the number of random bits and queries made by the verifier by a constant factor only. Thus if we define $\mathrm{PCP}_\varepsilon(\cdot,\cdot)$ as the class of languages that can be recognized by $(\cdot,\cdot)$-restricted verifiers that have an error probability of at most ε, we obtain

$$\mathrm{PCP}(\log n, 1) = \mathrm{PCP}_{1/2}(\log n, 1) = \mathrm{PCP}_\varepsilon(\log n, 1) \quad \forall \text{ constants } \varepsilon > 0.$$

While the error probability can be reduced to an arbitrarily small constant, the number of queries made by the $(\log n, 1)$-verifier must be at least 3, as the following result shows. (Here we use the notation $\mathrm{PCP}(., queries = .)$ to express that the constant in the $\mathcal{O}$-term for the number of queries is 1). For the adaptive version of this result see Exercise 4.1.)

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

Proposition 4.4. $\forall \varepsilon > 0 \quad \mathrm{PCP}_{\varepsilon}(\log n, queries = 2) = \mathcal{P}$

Proof. Clearly $\mathcal{P} \subseteq \mathrm{PCP}_{\varepsilon}(\log n, queries = 2)$ thus we only have to prove the other inclusion.

Let L be any language in $\mathrm{PCP}_{\varepsilon}(\log n, queries = 2)$ and let x be any input for which we want to decide in polynomial time whether $x \in L$ or not.

For each of the $2^{\mathcal{O}(\log n)}$ random strings we now can simulate the $(\log n, queries = 2)$-restricted verifier to see which two positions it would have queried from the proof and for what values of the queried bits the verifier would have accepted the input. For the ith random string r_i let b_{i1} and b_{i2} be the two bits queried by the verifier on input x and random string r_i. Now we can express by a 2SAT-formula in the two variables b_{i1} and b_{i2} whether the verifier would accept the input x. The verifier will accept the input x if and only if all the 2SAT-formulae obtained in this way from all the random strings r_i can be satisfied simultaneously. Thus we have reduced the problem of deciding whether $x \in L$ to a 2SAT-problem which can be solved in polynomial time. ∎

The number of queries needed by the verifier in the original proof of the PCP-Theorem of Arora et al. is about 10^4. This number has been reduced in a sequence of papers up to the current record due to Bellare, Goldreich and Sudan [BGS95] who proved that 11 queries to the proof suffice for the verifier to achieve an error probability of $1/2$:

$$\mathcal{NP} = \mathrm{PCP}(\log n, queries = 11)$$

However, as we shall see later, to obtain tight non-approximability results 11 queries are still too much. Even if one could proof that three queries suffice, this would not be strong enough to yield the desired non-approximability results. It turned out that instead of counting the number of queries needed by the verifier the right way of measuring the query complexity is expressed in so called *amortized free bits*. We will give the precise definition of this notion in Section 4.7 where it will become clear why amortized free bits are the right way to measure the query complexity of a verifier when one is interested in getting tight non-approximability results. While Proposition 4.4 shows that at least 3 bits have to be queried, unless $\mathcal{P}$=$\mathcal{NP}$, it was shown in a sequence of papers that the number of amortized free bits queried by the verifier can not only be smaller than 2, but even arbitrarily small. This is roughly the result of Håstad [Hås97a] that we mentioned in the introduction and allows to prove that MAXCLIQUE cannot be approximated up to a factor of $n^{1-\varepsilon}$ for arbitrarily small $\varepsilon > 0$.

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

4.4 Non-Approximability of $\mathcal{APX}$-Complete Problems

Arora, Lund, Motwani, Sudan and Szegedy [ALM+92] not only proved the PCP-Theorem but at the same time they also proved as a consequence that no $\mathcal{APX}$-complete problem has a PTAS.

Theorem 4.5. *Unless $\mathcal{P}=\mathcal{NP}$, no $\mathcal{APX}$-complete problem has a PTAS.*

Proof. We will show that the existence of a PTAS for MAX3SAT implies $\mathcal{P} = \mathcal{NP}$. Since MAX3SAT is $\mathcal{APX}$-complete (see Thereom 1.33) this proves the theorem.

Let L be an arbitrary language from $\mathcal{NP}$ and let V be the $(\log n, 1)$-restricted verifier for L whose existence is guaranteed by the PCP-Theorem.

For any input x we will use the verifier V to construct a 3SAT instance S_x such that S_x is satisfiable if and only if x is an element of L. Moreover, if x does not belong to L then at most some constant fraction of the clauses in S_x can simultaneously be satisfied. Therefore a PTAS for MAX3SAT could be used to distinguish between these two cases and the language L could be recognized in polynomial time, i.e., we would have $\mathcal{P} = \mathcal{NP}$.

We now describe how to construct the 3SAT instance S_x for a given input x using the verifier V. We interpret the proof queried by V as a sequence $x_1, x_2, x_3, \ldots$ of bits where the ith bit of the proof is represented by the variable x_i. For a given random string τ the verifier V will query $q = \mathcal{O}(1)$ bits from the proof which we will denote as $b_{\tau 1}, b_{\tau 2}, \ldots, b_{\tau q}$. It will accept the input x for the random string τ, if the bits $b_{\tau 1}, b_{\tau 2}, \ldots, b_{\tau q}$ have the correct values. Therefore we can construct a q-SAT formula F_τ that contains at most 2^q clauses such that F_τ is satisfiable if and only if there exists a proof π such that the verifier V accepts input x on random string τ. This q-SAT formula F_τ can be transformed into an equivalent 3SAT formula F'_τ containing at most $q \cdot 2^q$ clauses and possibly some new variables. We define S_x to be the conjunction of all formulae F'_τ for all possible random strings τ.

If x is an element of L then by definition of a restricted verifier there exists a proof π such that V accepts x for every random string τ. Thus the formula S_x is satisfiable.

If x is not an element of L then for every proof π the verifier V accepts x for at most $1/2$ of all possible random strings τ. This means that at most $1/2$ of the formulae F'_τ are simultaneously satisfiable. Since every F'_τ consists of at most $q \cdot 2^q$ clauses we get that at least a $\frac{1}{2 \cdot q \cdot 2^q}$ fraction of the clauses of S_x are not satisfiable.

The existence of a PTAS for MAX3SAT therefore would allow to distinguish between these two cases and thus it would be possible to recognize every language in $\mathcal{NP}$ in polynomial time. ∎

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

From this proof and the $\mathcal{NP} = \text{PCP}(\log n, queries = 11)$-result stated in Section 4.3, we immediately get the following corollary.

Corollary 4.6. *Unless $\mathcal{P} = \mathcal{NP}$ no polynomial time approximation algorithm for* MAX3SAT *can have a performance guarantee of* $45056/45055 = 1.000022\ldots$

Proof. The proof of Theorem 4.5 has shown that there exists no polynomial time approximation algorithm for MAX3SAT with a performance guarantee of $1/(1 - \frac{1}{2 \cdot q \cdot 2^q})$, unless $\mathcal{P} = \mathcal{NP}$. Setting $q = 11$ we get the claimed result. ■

The constant we achieved in this corollary of course is far from being optimal. The first reasonable constant for the non-approximability of MAX3SAT was obtained by Bellare, Goldwasser, Lund and Russell [BGLR93]. They obtained $94/93 = 1.0107\ldots$ which was improved by Bellare and Sudan [BS94] to $66/65 = 1.0153\ldots$ and by Bellare, Goldreich and Sudan [BGS95] to $27/26 = 1.0384\ldots$ until very recently Håstad[Hås97b] improved this to the best possible result of $8/7 - \varepsilon = 1.142\ldots$ (see Chapter 7).

The proof of Theorem 4.5 shows that there exists a constant $\varepsilon > 0$ such that the following promise problem, called ROBE3SAT, is $\mathcal{NP}$-hard:

ROBE3SAT

Instance: A 3SAT formula Φ such that Φ is either satisfiable or at least an ε-fraction of the clauses of Φ is not satisfiable

Question: Is Φ satisfiable ?

If we use 3SAT as the language L in the proof of Theorem 4.5 we see that instances of the ordinary 3SAT problem can be transformed in polynomial time into instances of ROBE3SAT such that satisfiable instances are transformed into satisfiable instances and unsatisfiable instances are transformed into instances where at least an ε-fraction of the clauses is unsatisfiable.

In Chapter 5 it will be shown (see Theorem 5.54) that for the verifier constructed in the proof of the PCP-Theorem we have the following property: given a proof π that is accepted with probability larger than the soundness probability for an input x, one can construct in polynomial time a new proof π' that is accepted with probability 1 for input x.

If we apply this result to the instances of ROBE3SAT that are constructed from ordinary 3SAT instances as described in the proof of Theorem 4.5 we get the following result.

Corollary 4.7. *There exists a polynomial time computable function g that maps* 3SAT *instances to* ROBE3SAT *instances and has the following properties:*

– *if x is a satisfiable* 3SAT *instance then $g(x)$ is.*

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

- *if x is an unsatisfiable* 3Sat *instance then at most an $1-\varepsilon$ fraction of the clauses of $g(x)$ are simultaneously satisfiable.*
- *given an assignment to $g(x)$ that satisfies more than an $1-\varepsilon$ fraction of the clauses of $g(x)$ one can construct in polynomial time a satisfying assignment for x.*

4.5 Expanders and the Hardness of Approximating MaxE3Sat-b

In this section we will show that the $\mathcal{NP}$-hardness of RobE3Sat also holds for the special case of RobE3Sat-b, i.e., for 3Sat-formulae where each clause contains exactly three literals and each variable appears at most b times. This result will be used in Chapter 7 and Chapter 10.

For a constant k, a k-regular (multi-)graph $G=(V,E)$ is called an *expander*, if for all sets $S \subset V$ with $|S| \leqslant |V|/2$ there are at least $|S|$ edges connecting S and $V-S$. The next lemma shows that for every sufficiently large n, there exist sparse expanders.

Lemma 4.8. *For every sufficiently large n and any constant $k \geqslant 5$ there exist $4k$-regular expanders.*

Proof. We construct a bipartite graph on sets $A=B=\{1,\ldots,n\}$ by choosing k random permutations $\pi_1,\ldots,\pi_k$ and connecting vertex i in A with vertices $\pi_1(i),\ldots,\pi_k(i)$ in B. This way we get a k-regular bipartite graph on n vertices. We claim that there exist permutations $\pi_1,\ldots,\pi_k$ such that whenever we choose a set $S \subset A$ with $|S| < n/2$ then there are at least $3|S|/2$ vertices in B that are adjacent to some vertex in S.

Let S be a subset of A with $t := |S| \leqslant n/2$ and T be the set containing all vertices in B that are adjacent to some vertex in S, such that $m := |T| = \lfloor (3|S|-1)/2 \rfloor$. We will call such pairs (S,T) *bad*. The probability that for randomly chosen permutations $\pi_1,\ldots,\pi_k$ and fixed sets S and T these sets form a bad pair equals

$$\left(\binom{m}{t} \frac{t!(n-t)!}{n!} \right)^k .$$

If we sum this up over all possible choices for S and T we see that the probability that there exists a bad pair is bounded by

$$\sum_{t \leqslant \frac{n}{2}} \binom{n}{m} \left(\binom{n}{t} \frac{m!(n-t)!}{n!(m-t)!} \right)^k .$$

For $1 \leqslant t < n/3$ it is easily verified that the function

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

$$f(t) := \binom{n}{t}\binom{n}{m}\frac{m!(n-t)!}{n!(m-t)!}$$

satisfies $f(t) > f(t+1)$, so the maximum of f in this range is attained at $t = 1$. For $n/3 \leqslant t \leqslant n/2$ the expression $\frac{m!(n-t)!}{(m-t)!}$ reaches its maximum for $t = n/3$ or $t = n/2$. Now a simple computation using Stirling's formula shows that $\frac{n}{2}(f(1) + f(n/3) + f(n/2))$ tends to zero for $k \geqslant 5$ and n going to infinity.

Therefore there exist k-regular bipartite graphs with n vertices on each side such that every set S from the left side with $|S| \leqslant n/2$ has at least $3/2|S|$ neighbors on the right side. Now, if we identify vertices with the same number from both sides and duplicate each edge, we obtain a $4k$-regular (multi-)graph on n vertices such that from every set S of vertices of size at most $n/2$ there are at least $|S|$ edges leaving the set S. ∎

While the above lemma shows the *existence* of expanders only, it can be shown that expanders can be *constructed* explicitly in polynomial time. See for example [Mar75] or [GG81] for such constructions.

With the help of expanders we are now able to prove the desired hardness result for RobE3Sat-b.

Lemma 4.9. *There exists a constant b such that* RobE3Sat-b *is $\mathcal{NP}$-hard.*

Proof. We prove this by reducing from RobE3Sat. Given a 3Sat formula F with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$, we replace each of the say k_i occurrences of the variable x_i by new variables $y_{i,1}, \ldots, y_{i,k_i}$ and choose an expander G_i with $y_{i,1}, \ldots, y_{i,k_i}$ as its vertices. Now we add the clauses $(y_{i,a} \vee \overline{y}_{i,b})$ and $(\overline{y}_{i,a} \vee y_{i,b})$ whenever $y_{i,a}$ and $y_{i,b}$ are connected by an edge in G_i. We call this new 3Sat formula F'. Since the expanders G_i have constant degree, each variable in F' appears only a constant number of times. Moreover, F' still has $\mathcal{O}(m)$ many clauses. Whenever we have an assignment to F' we may assume that for all i the variables $y_{i,1}, \ldots, y_{i,k_i}$ have the same value. If this was not the case, we simply could change the values of $y_{i,1}, \ldots, y_{i,k_i}$ to the value of the majority of these variables. This way, we may loose up to $k_i/2$ satisfied clauses, but at the same time we gain at least $k_i/2$. This follows from the fact that G_i is an expander and therefore every set S of vertices of size at most $k_i/2$ has at least $|S|$ edges leaving it. Each of these edges yields an unsatisfied clause before changing the values of $y_{i,1}, \ldots, y_{i,k_i}$ to the value of their majority. Therefore a solution to F' can be used to define a solution to F and both formulae have the same number of unsatisfiable clauses. ∎

Corollary 4.10. *There exist constants $\delta > 0$ and b such that no polynomial time algorithm can approximate* MaxE3Sat-b *up to a factor of $1 + \delta$, unless $\mathcal{P}$=$\mathcal{NP}$.*

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

Proof. Suppose there exists a polynomial time $1+\delta$ approximation algorithm for MaxE3Sat-b for all $\delta > 0$. Given an instance of RobE3Sat-b such that either all clauses are satisfiable or at least an ε-fraction of the clauses is not satisfiable, this algorithm with $\delta < \varepsilon/(1-\varepsilon)$ could be used to distinguish between these two cases in polynomial time. This implies $\mathcal{P}=\mathcal{NP}$ because of Lemma 4.9. ∎

4.6 Non-Approximability of MaxClique

A *clique* in a graph is a set of pairwise adjacent vertices. The problem Clique is defined as follows.

Clique
Instance: Given a graph G and an integer k
Question: Is there a clique of size $\geqslant k$ in G ?

The corresponding optimization problem is called MaxClique.

MaxClique
Instance: Given a graph G
Problem: What is the size of a largest clique in G ?

While it is a classical result due to Karp [Kar72] that Clique is $\mathcal{NP}$-complete there was not any non-approximability result known for the problem MaxClique up to the year 1991 when Feige, Goldwasser, Lovász, Safra and Szegedy [FGL+91] observed a connection between PCPs and MaxClique. The only result known long before this is the fact that MaxClique is a *self-improving* problem:

Proposition 4.11. *If for any constant c there is a c-approximation-algorithm for* MaxClique*, then there also exists a $\sqrt{c}$-approximation algorithm for* MaxClique.

Proof. This result follows immediately from the fact that the product of a graph G with itself (replace each vertex of G by a copy of G and join two such copies completely if the corresponding vertices in G are adjacent) yields a new graph G' whose maximum clique size is the square of the size of a maximum clique in G. Thus a c-approximation algorithm for G' can be used to obtain a $\sqrt{c}$-approximation algorithm for G. ∎

This self-improving property implies that if there exists *any* constant factor approximation algorithm for MaxClique then there even exists a PTAS for this problem. As the best known approximation algorithm for MaxClique due to Boppana and Halldórsson [BH92] has a performance guarantee of $\mathcal{O}(n/\log^2 n)$,

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

the existence of a PTAS for MAXCLIQUE was assumed to be extremely unlikely but could not be ruled out before 1991.

In this section we will see how the PCP-Theorem implies the nonexistence of a polynomial time n^{ε}-approximation algorithm for MAXCLIQUE for some $\varepsilon > 0$, while in the next section we will show that even an $n^{1-\delta}$ approximation algorithm does not exist for this problem for arbitrarily small δ, unless $\mathcal{NP}=\mathcal{ZPP}$. We start by showing that no polynomial time constant factor approximation algorithm for MAXCLIQUE can exist.

Proposition 4.12. *Unless $\mathcal{P}=\mathcal{NP}$, no polynomial time constant factor approximation algorithm for* MAXCLIQUE *can exist.*

Proof. We use the standard reduction from 3SAT to CLIQUE to prove this result.

For a given 3SAT-formula F with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$ we construct a graph G on $3m$ vertices $(i,j), i = 1, \ldots, m; j = 1, 2, 3$ as follows. The vertices (i,j) and (i',j') are connected by an edge if and only if $i \neq i'$ and the jth literal in clause i is not the negation of the j'th literal in clause i'.

If there exists a clique in G of size k then it contains at most one literal from each clause, and it contains no two literals that are the negation of each other. Therefore, by setting all literals corresponding to vertices of this clique to *true* one gets a truth assignment for F that satisfies at least k of its clauses. On the other hand, given a truth assignment for F that satisfies k of the clauses, one gets a clique of size k in G by simply selecting from each satisfied clause one literal that evaluates to *true* in this assignment.

Thus we have shown that the graph G has a clique of size k if and only if there exists a truth assignment for F that satisfies k of its clauses. This shows that a PTAS for MAXCLIQUE cannot exist as otherwise we would also get a PTAS for MAX3SAT which is ruled out by Theorem 4.5. Proposition 4.11 now implies that for no constant c a c-approximation algorithm for MAXCLIQUE can exist. ■

To prove better non-approximability results for MAXCLIQUE, especially for proving the n^{ε} non-approximability we have to make a more direct use of the PCP-Theorem. To start with we first present a reduction from 3SAT to CLIQUE that is slightly different from the one used in the proof of Proposition 4.12 and was used by Papadimitriou and Steiglitz [PS82] to prove the $\mathcal{NP}$-completeness of CLIQUE.

Proof. (Second proof for Proposition 4.12)

Again we are given a 3SAT-formula F with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$. The idea this time is that for each clause we want to list all truth assignments that make this clause true. Instead of listing this exponential number of assignments, we only list 7 *partial* truth assignments for each clause.

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

A partial truth assignment assigns the values *true* and *false* to certain variables only; the rest of the variables has the value '$\cdot$', meaning that the value is undefined. As an example, a partial truth assignment for variables $x_1, \ldots, x_9$ might look like $\cdot 10 \cdot 0 \cdot \cdot 01$. We say that two different truth assignments t and t' are *compatible*, if for all variables x for which $t(x) \neq \cdot$ and $t'(x) \neq \cdot$ we have $t(x) = t'(x)$. For each clause C_i there are exactly 7 satisfying partial truth assignments with values defined only on the three variables appearing in C_i (we assume here without loss of generality that every clause contains exactly three different variables). We construct for each of the m clauses of F these 7 partial truth assignments and take them as the vertices of our graph G. Two vertices in G are connected if the corresponding truth assignments are compatible.

First note that no two partial truth assignments corresponding to the same clause of F can be compatible and therefore G is an m-partite graph. Now if there exists a clique of size k in G then this means that there is a set of k pairwise compatible partial truth assignments for k different clauses of F. Thus there exists one truth assignment that satisfies all these k clauses simultaneously. On the other hand, if there is a truth assignment for F that satisfies k of its clauses, then there is one partial truth assignment for each of these clauses that is compatible to this truth assignment and therefore these k partial truth assignments are pairwise compatible yielding a clique of size k in G. ■

We will now see – as was discovered by Feige, Goldwasser, Lovász, Safra and Szegedy [FGL+91] – that the reduction of Papadimitriou and Steiglitz applied to the PCP-result will achieve the n^{ε} non-approximability result for CLIQUE. As a first step we will prove once more Proposition 4.12.

Proof. (Third proof for Proposition 4.12)

Let L be an $\mathcal{NP}$-complete language and V be its $(\log n, 1)$-restricted verifier whose existence is guaranteed by the PCP-Theorem. Let $r(n) = \mathcal{O}(\log n)$ and $q(n) = \mathcal{O}(1)$ be the number of random bits respectively query bits used by the verifier V. Now for an input x we construct a graph G_x in an analogous way as Papadimitriou and Steiglitz did in their reduction from 3SAT to CLIQUE as described in the second proof of Proposition 4.12. The role of a clause appearing in the 3SAT formula is now taken by a random string read by the verifier and the 3 variables appearing in a given clause correspond to the $q(n)$ positions queried from the proof by the verifier.

For each of the possible $2^{r(n)}$ random strings we list all of the at most $2^{q(n)}$ partial proofs (i.e., assignments of 0 and 1 to the positions queried by the verifier for the given random string, and assignment of '$\cdot$' to all other positions) that will make the verifier V accept the input x. All these partial proofs are vertices in our graph G_x and we connect two such vertices if they are compatible (as defined above). The graph G_x has at most $2^{r(n)+q(n)}$ vertices and since for two given partial proofs it can be decided in polynomial time whether they are compatible, the graph can be constructed in polynomial time.

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

For a fixed proof π any two vertices of G_x that are compatible with π are adjacent. Therefore, if there exists a proof π such that the verifier V accepts the input x for k different random strings, then the graph G_x contains a clique of size k.

If on the other hand, the graph G_x contains a clique of size k, then the k partial proofs corresponding to the vertices of the clique are pairwise compatible and as no two partial proofs that correspond to the same random string are compatible with each other, there must exist a proof π such that the verifier accepts the input x for k different random strings.

Thus we have shown that the size of a maximum clique in G_x equals the maximum number of random strings for which the verifier accepts a proof π, where the maximum is taken over all proofs π.

Now if $x \in L$ then by the definition of PCP there exists a proof π such that the verifier accepts for all possible random strings. Thus in this case $\omega(G_x) = 2^{r(n)}$.

If $x \notin L$ then by the definition of PCP for each proof π the verifier accepts x for at most $1/2$ of the random strings. Therefore we have $\omega(G_x) \leqslant \frac{1}{2}2^{r(n)}$ in this case.

Now a 2-approximation algorithm for MaxClique could be used to recognize the $\mathcal{NP}$-complete language L in polynomial time. ■

For the reduction we used in the above proof, the non-approximability factor we obtain for MaxClique solely depends on the error probability of the verifier. We have already seen, that this error probability can be made arbitrarily, but constantly small as we know that $\mathcal{NP} = PCP_{\varepsilon}(\log n, 1)$ for all $\varepsilon > 0$. To obtain an n^{ε} non-approximability result for MaxClique we had to reduce the error probability of the verifier to $n^{-\varepsilon}$. Clearly this can be done by running the $(\log n, 1)$-restricted verifier for $\mathcal{O}(\log n)$ independent random strings; however this would result in a total number of $\mathcal{O}(\log^2 n)$ random bits needed by the verifier which results in a graph that can no longer be constructed in polynomial time.

The idea here now is that instead of using truly random bits one can make use of so called *pseudo random bits* that can be generated by performing a random walk on an expander graph. It can be shown that by this method one can generate $\alpha \log n$ pseudo random strings of length $\mathcal{O}(\log n)$ by using only $c \cdot \alpha \log n$ truly random bits (for more details on this see for example [HPS94]). Thus, starting with an $(\log n, 1)$-verifier V that has error probability of $1/2$ we can construct a new verifier V' that simulates the verifier V $\alpha \log n$ times. If q is the number of bits queried by the verifier V, then we get for the new verifier V' :

error probability	:	$n^{-\alpha}$
# random bits	:	$c \cdot \alpha \log n$
# query bits	:	$q \cdot \alpha \log n$

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

Now if we use this verifier to construct for an input x a graph G_x as described above, we get that the clique number of G_x cannot be approximated up to n^α for arbitrarily large but constant α.

The graph G_x has

$$N := 2^{c\cdot\alpha\log n + q\cdot\alpha\log n} = n^{c\cdot\alpha+q\cdot\alpha}$$

vertices. Thus we get:

$$n^{-\alpha} = N^{-\frac{\alpha}{c\alpha+q\alpha}} = N^{-\frac{1}{c+q}}$$

As c and q are constants, we have shown:

Theorem 4.13. *Unless $\mathcal{P}$=$\mathcal{NP}$, there exists a constant $\varepsilon > 0$ such that no n^ε approximation algorithm for* MaxClique *can exist.*

4.7 Improved Non-Approximability Results for MaxClique

In the last section we have seen, that MaxClique cannot be approximated up to n^ε for some constant ε. Here we now want to see how large this ε can be.

The value of ε was $\varepsilon = 1/(c+q)$ where c was a constant that came in from the generation of pseudo random bits and q is the number of queries made by the $(\log n, 1)$-restricted verifier. Thus to achieve a small value for ε we have to try to minimize these two constants. It can be shown that by using Ramanujan-expanders due to Lubotzky, Phillips and Sarnak [LPS86] for the generation of pseudo random bits, the constant c can almost achieve the value 2. As we already know that 11 queries are enough for an $(\log n, 1)$-restricted verifier, this shows that we can choose $\varepsilon = 0.076$.

From this value for ε up to the ultimate result due to Håstad [Hås97a] showing that ε can be chosen arbitrarily close to 1, there was a long sequence of improvements which is surveyed in Table 4.1.

Friedman [Fri91] has shown that the Ramanujan-expanders of Lubotzky, Phillips and Sarnak are best possible, meaning that the constant c must have at least the value 2. On the other hand we know from Proposition 4.4 that q must be at least 3. Thus to get values of ε that are larger than $1/5$ we need some new ideas.

First we note, that in the construction of the graph G_x in the third proof of Proposition 4.12 we listed for each of the $2^{r(n)}$ random bits all partial proofs that made the verifier V accept the input x. As V queries at most $q(n)$ bits there can be at most $2^{q(n)}$ partial proofs for which V accepts x. However, a close look at the proof of the PCP-Theorem reveals, that for a fixed random string there are usually much less than $2^{q(n)}$ accepted partial proofs. The reason for

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

Authors	*Factor*	*Assumption*
Feige, Goldwasser, Lovász, Safra, Szegedy 91	$\exists \varepsilon > 0, 2^{\log^{1-\varepsilon} n}$	$\mathcal{NP} \neq \tilde{\mathcal{P}}$
Arora, Safra 92	$\exists \varepsilon > 0, 2^{\log^{1-\varepsilon} n}$	$\mathcal{NP} \neq \mathcal{P}$
Arora, Lund, Motwani, Sudan, Szegedy 92	$n^{1/10000}$	$\mathcal{NP} \neq \mathcal{P}$
Bellare, Goldwasser, Lund, Russell 93	$n^{1/30}$ $n^{1/25}$	$\mathcal{NP} \neq$ co-$\mathcal{RP}$ $\mathcal{NP} \neq$ co-$\mathcal{R}\tilde{\mathcal{P}}$
Feige, Kilian 94	$n^{1/15}$	$\mathcal{NP} \neq$ co-$\mathcal{RP}$
Bellare, Sudan 94	$\forall \varepsilon, n^{1/6-\varepsilon}$ $\forall \varepsilon, n^{1/5-\varepsilon}$ $\forall \varepsilon, n^{1/4-\varepsilon}$	$\mathcal{NP} \neq \mathcal{P}$ $\mathcal{NP} \neq$ co-$\mathcal{RP}$ $\mathcal{NP} \neq$ co-$\mathcal{R}\tilde{\mathcal{P}}$
Bellare, Goldreich, Sudan 95	$\forall \varepsilon, n^{1/4-\varepsilon}$ $\forall \varepsilon, n^{1/3-\varepsilon}$	$\mathcal{NP} \neq \mathcal{P}$ $\mathcal{NP} \neq$ co-$\mathcal{RP}$
Håstad 96	$\forall \varepsilon, n^{1/2-\varepsilon}$	$\mathcal{NP} \neq$ co-$\mathcal{RP}$
Håstad 96	$\forall \varepsilon, n^{1-\varepsilon}$	$\mathcal{NP} \neq$ co-$\mathcal{RP}$

Table 4.1. Non-approximability results for MAXCLIQUE.

this is, that the verifier will often query some bits, say b_1, b_2, b_3 and then it will test whether these queried bits satisfy a certain relation, say $g(b_1)+g(b_2) = g(b_3)$ for some function g. If this relation is not satisfied, then V will not accept the input x. For this example it follows, that instead of 8 possible answers for the bits b_1, b_2, b_3 there can be at most 4 answers for which V accepts the input x. Roughly speaking, instead of counting the number of bits queried by the verifier from the proof, it is only of interest, how many of these queried bits have no predetermined value. These bits are called *free bits* and denoted by f.

More precisely the number f of *free bits* queried by a verifier is defined as

$$f := \log(\max_{\tau, x} \# \text{ partial proofs for which } V \text{ accepts } x).$$

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

Following [BS94] we define the class FPCP as the free bit variant of PCP, i.e., the class of languages where we measure the number of free query bits instead of query bits.

The idea of free bits appears for the first time in the paper of Feige and Kilian [FK94] who proved that MAXCLIQUE cannot be approximated up to $n^{1/15}$ unless $\mathcal{NP}$=co-$\mathcal{RP}$. The name 'free bits' was invented by Bellare and Sudan [BS94].

Thus, to improve the non-approximability factor for MAXCLIQUE, which we now know is $\varepsilon = 1/(c+f)$ one carefully has to look at the proof of the PCP-Theorem to see how many free bits are needed. Bellare, Goldreich and Sudan [BGS95] have shown in a result similar to Proposition 4.4 that at least 2 free bits are needed.

Proposition 4.14. $\forall \varepsilon > 0 \quad \mathrm{FPCP}_{\varepsilon}(\log n,\ \textit{free bits} = 1) = \mathcal{P}$

On the other hand they also showed that 2 free bits suffice for a $(\log n, 1)$-restricted verifier to recognize any $\mathcal{NP}$-language.

Theorem 4.15. $\mathcal{NP} \subseteq \mathrm{FPCP}_{0.794}(\log n,\ \textit{free bits} = 2)$

They also proved the best known result for error probability $1/2$.

Theorem 4.16. $\mathcal{NP} \subseteq \mathrm{FPCP}_{1/2}(\log n,\ \textit{free bits} = 7)$

Still these results only yield that we cannot get a polynomial time $n^{1/4}$-approximation algorithm for MAXCLIQUE. Before further improving on the query complexity we will see how the constant c in the expression for ε can be decreased to 1 by using a more efficient method of generating pseudo random bits due to Zuckerman [Zuc93].

An (m, n, d)-*amplification scheme* is a bipartite graph $G = (A \cup B, E)$ with $|A| = m$ and $|B| = n$ such that every vertex in A has degree d. We construct (m, n, d)-amplification schemes uniformly at random by choosing for each vertex in A uniformly at random d elements of B as neighbors. We are interested in amplification schemes that satisfy a certain expansion property.

An (m, n, d, a, b)-*disperser* is a bipartite graph $G = (A \cup B, E)$ with m vertices on the left and n vertices on the right such that every vertex on the left has degree d and each subset of size a on the left has at least b neighbors. The following result shows that for certain parameter sets (m, n, d, a, b)-dispersers can randomly be constructed in a very simple way.

Lemma 4.17. *The probability that a uniformly at random chosen* $(2^R, 2^r, R+2)$*-amplification scheme is a* $(2^R, 2^r, R+2, 2^r, 2^{r-1})$*-disperser is at least* $1 - 2^{-2^r}$.

Proof. For $S \subseteq 2^R$ and $T \subseteq 2^r$ let $A_{S,T}$ be the event that all neighbors of S are in T. Then the probability that the randomly chosen $(2^R, 2^r, R+2)$-amplification scheme is not the desired disperser equals

$$\begin{aligned}
\text{Prob}[\bigcup_{\substack{|S|=2^r \\ |T|=2^{r-1}-1}} A_{S,T}] &\leqslant \sum_{\substack{|S|=2^r \\ |T|=2^{r-1}-1}} \text{Prob}[A_{S,T}] \\
&= \binom{2^R}{2^r}\binom{2^r}{2^{r-1}-1}\left(\frac{2^{r-1}-1}{2^r}\right)^{(R+2)2^r} \\
&< 2^{R2^r}2^{2^r}2^{-(R+2)2^r} \\
&= 2^{-2^r}
\end{aligned}$$

■

We will use these $(2^R, 2^r, R+2, 2^r, 2^{r-1})$-dispersers to generate $R+2$ pseudo random strings of length r in a very simple way: We simply choose a vertex from the left side and take all its $R+2$ neighbors as pseudo random strings. The following result shows that by doing so we can reduce the constant c to 1.

Theorem 4.18. *Unless $\mathcal{NP}=\mathcal{ZPP}$ no polynomial time algorithm can achieve an approximation factor of $n^{\frac{1}{1+f}-\varepsilon}$ for* MaxClique *for arbitrarily small ε.*

Proof. Let V be a verifier for recognizing a language L that uses $r(n)$ random bits, queries f free bits and achieves an error probability of $1/2$. We will construct a verifier V' now as follows.

The verifier V' first uniformly at random chooses a $(2^R, 2^{r(n)}, R+2)$-amplification scheme which by Lemma 4.17 is with very high probability a $(2^R, 2^{r(n)}, R+2, 2^{r(n)}, 2^{r(n)-1})$-disperser. Now V' randomly selects a vertex from the left side of the disperser and uses its $R+2$ neighbors as random strings of length $r(n)$. For each of these $R+2$ random strings the verifier V' simulates the verifier V for input x. It accepts x, if and only if V accepts x for all $R+2$ runs.

The verifier V' uses R random bits and its free bit complexity is $(R+2)f$. Thus the graph G_x constructed for input x by the same construction as described in the third proof of Proposition 4.12 has $N := 2^{R+(R+2)f}$ vertices.

If $x \in L$ then there exists a proof π such that V accepts input x for all random strings and therefore V' accepts for all 2^R random strings. Thus the graph G_x has a clique of size 2^R.

If $x \notin L$ then we claim that G_x has a clique of size at most $2^{r(n)}$. Assume this is not the case, i.e., there exist $p > 2^{r(n)}$ random strings for which V' accepts

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

input x. Since V accepts input x for less than $\frac{1}{2}2^{r(n)}$ random strings, this means that there are $p > 2^{r(n)}$ vertices on the left of the disperser whose neighborhood contains at most $2^{r(n)-1} - 1$ vertices. This contradicts the definition of a $(2^R, 2^{r(n)}, R+2, 2^{r(n)}, 2^{r(n)-1})$-disperser.

Thus we cannot distinguish in polynomial time whether there is a clique of size 2^R or whether every clique has size at most $2^{r(n)}$, i.e., MAXCLIQUE cannot be approximated up to a factor of $2^{R-r(n)}$.

Now we have:

$$\begin{aligned} N &= 2^{R+(R+2)f} \\ &= 2^{R(1+f)} \cdot 2^{2f} \\ \Rightarrow \quad 2^R &= \left(\frac{N}{2^{2f}}\right)^{\frac{1}{1+f}} \end{aligned}$$

If we choose $R = \alpha \log n$ we get for $\alpha \to \infty$ that MAXCLIQUE cannot be approximated up to $N^{\frac{1}{1+f}-\varepsilon}$ for arbitrarily small ε, unless $\mathcal{NP}$=co-$\mathcal{RP}$ (we used a randomized construction for the disperser). ■

Since by Theorem 4.15 we know that we can set $f = 2$ we get that no $n^{1/3-\varepsilon}$ approximation algorithm can exist for MAXCLIQUE. On the other hand we know that f must be larger than 1. Thus we need to refine the notion of query complexity once more to arrive at the final tight non-approximability result for MAXCLIQUE.

The observation here now is, that for our non-approximability results we only made use of the fact that there is a certain gap between the completeness and soundness probability of the verifier. So far we have assumed that the completeness probability is always 1; in this case the verifier is said to have *perfect completeness*. All the non-approximability results for MAXCLIQUE we have seen so far do not need perfect completeness. We already have observed the trade off between the error gap and the number of queries: we can square the error gap by just doubling the number of queries, i.e., if we want to enlarge the logarithm of the error gap by a factor of k, then we have to allow the verifier to use k times as many query bits. A careful analysis of the proof of Theorem 4.18 reveals that in fact the non-approximability-factor for MAXCLIQUE does not just depend on f but on the ratio of f and the logarithm of the error gap.

This motivates the definition of so called *amortized free bit complexity* . If a verifier has completeness probability c and soundness probability s and has free bit complexity f, then its amortized free bit complexity $\overline{f}$ is defined as

$$\overline{f} := f/\log(c/s).$$

We define the class $\overline{\text{FPCP}}$ as the amortized free bit complexity variant of PCP. In the proof of Theorem 4.18 we used an error gap of 2. Bellare and Sudan

This paper appeared in: Lectures on Proof Verification and Approximation Algorithms, 63-82, LNCS 1367, 1998

[BS94] have shown that the same result as Theorem 4.18 can be proved in terms of amortized free bit complexity.

Theorem 4.19. *Unless $\mathcal{NP}=\mathcal{ZPP}$ no polynomial time algorithm can achieve an approximation factor of $n^{\frac{1}{1+7}-\varepsilon}$ for* MaxClique *for arbitrarily small ε.*

Håstad [Hås97a] has shown (see Chapter 9) that $\mathcal{NP} \subseteq \overline{\text{FPCP}}(\log n,$ *amortized free bits* $= 0)$. Together with Theorem 4.19 this yields the desired non-approximability result for MaxClique.

Theorem 4.20. *Unless $\mathcal{NP}=\mathcal{ZPP}$ no polynomial time algorithm can achieve an approximation factor of $n^{1-\varepsilon}$ for* MaxClique *for arbitrarily small ε.*

Exercises

Exercise 4.1. Prove Proposition 4.4 for the adaptive case.

Exercise 4.2. $\text{PCP}(1, \log n) = ?$

Exercise 4.3. What non-approximability result for MaxClique can be obtained by rerunning the $(\log n, 1)$-restricted verifier $\mathcal{O}(\log n)$ times, i.e., without making use of pseudo random bits ?

Exercise 4.4. $\text{FPCP}(\log n,$ *free bits* $= 1) = \mathcal{P}$

Exercise 4.5. Why did we not have to take into account in the proof of Theorem 4.18 the random bits needed to create the disperser ?

Exercise 4.6. Show that the PCP-Theorem implies that the average number of queries needed by a $(\log n, 1)$-restricted verifier can be made arbitrarily small, while increasing the error probability.