Introduction to Complexity Theory*
Lecture 12: Probabilistically Checkable Proof Systems

Notes taken by Alon Rosen and Vered Rosen

June 21, 1999

Summary: In this lecture we introduce the notion of Probabilistically Checkable Proof (PCP) systems. We discuss some complexity measures involved, and describe the class of languages captured by corresponding PCP systems. We then demonstrate the alternative view of $\mathcal{NP}$ emerging from the PCP theorem, and use it in order to prove two non-approximability results for the problems $max3SAT$ and $maxCLIQUE$.

1 Introduction

Loosely speaking, a probabilistically checkable proof system (PCP) for a language consists of a probabilistic polynomial-time verifier having direct access to individual bits of a binary string. This string (called oracle) represents a proof, and typically will be accessed only partially by the verifier. Queries to the oracle are positions on the bit string and will be determined by the verifier's input and coin tosses (potentially, they might be determined by answers to previous queries as well). The verifier is supposed to decide whether a given input belongs to the language.

If the input belongs to the language, the requirement is that the verifier will always accept (i.e. given access to an adequate oracle). On the other hand, if the input does not belong to the language then the verifier will reject with probability at least $\frac{1}{2}$, no matter which oracle is used.

One can view PCP systems in terms of interactive proof systems. That is, one can think of the oracle string as being the prover and of the queries as being the messages sent to him by the verifier. In the PCP setting however, the prover is considered to be memoryless and thus cannot adjust his answers based on previous queries posed to him.

A more appealing interpretation is to view PCP systems as a possible way of generalizing $\mathcal{NP}$. Instead of conducting a polynomial-time computation upon receiving the entire proof (as in the case of $\mathcal{NP}$), the verifier is allowed to toss coins and query the proof only at locations of his choice. This either allows him to inspect very long proofs (looking at no more than polynomially many locations), or alternatively, look at very few bits of a possible proof.

Most surprisingly, PCP systems have been used to fully characterize the languages in $\mathcal{NP}$. This characterization has been found to be useful in connecting the hardness involved in the approximation of some *NP-hard* problems with the $\mathcal{P} \neq \mathcal{NP}$ question. In other words, very strong non-approximability results for various classical optimization problems have been established using PCP systems for $\mathcal{NP}$ languages.

---

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

2 The Definition

2.1 The basic model

In the definition of PCP systems we make use of the notion of a probabilistic oracle machine. In our setting, this will be a probabilistic Turing machine which, in addition to the usual features, will have direct access (counted as a single step) to individual bits of a binary string (the oracle). ¿From now on, we denote by $M^{\pi}(x)$ the output of machine M on input x, when given such oracle access to the binary string π.

Definition 1 (Probabilistically Checkable Proofs - PCP) *A probabilistic checkable proof system for a language L is a probabilistic polynomial-time oracle machine* (called verifier)*, denoted M, satisfying*

- *Completeness: For every $x \in L$ there exists an oracle π_x such that:*

$$\Pr\left[M^{\pi_x}(x) = 1\right] = 1$$

- *Soundness: For every $x \notin L$ and every oracle π:*

$$\Pr\left[M^{\pi}(x) = 1\right] \leq \frac{1}{2}$$

where the probability is taken over M's internal coin tosses.

2.2 Complexity Measures

When considering a randomized oracle machine, some complexity measures other than time may come into concern. A natural thing would be to count the number of queries made by the verifier. This number determines what is the portion of the proof being read by the verifier. Another concern would be to count the number of coins tossed by the randomized oracle machine. This in turn determines what is the total number of possible executions of the verifier (once an oracle is fixed).

It turns out that the class of languages captured by PCP systems varies greatly as the above mentioned resources of the verifier are changed. This motivates a quantitative refinement of the definition of PCP systems which captures the above discussed concept.

Definition 2 (Complexity Measures for PCP) *Let $r, q : N \to N$ be integer functions (in particular constant). The complexity class $\mathcal{PCP}(r(\cdot), q(\cdot))$ consists of languages having a probabilistic checkable proof system in which it holds that:*

- Randomness Complexity*: On input $x \in \{0,1\}^*$, the verifier makes at most $r(|x|)$ coin tosses.*
- Query Complexity*: On input $x \in \{0,1\}^*$, the verifier makes at most $q(|x|)$ queries.*

For sets of integer functions R and Q, we let

$$\mathcal{PCP}(R, Q) \stackrel{\text{def}}{=} \bigcup_{r \in R, q \in Q} \mathcal{PCP}(r(\cdot), q(\cdot))$$

In particular, we denote by poly the set of all integer functions bounded by a polynomial and by log the set of all integer functions bounded by a logarithmic function (e.g. $f \in \log$ iff $f(n) = O(\log n)$). From now on, whenever referring to a PCP system, we will also specify its corresponding complexity measures.

2.3 Some Observations

- The definition of PCP involves binary queries to the oracle (which is itself a binary string). These queries specify locations on the string whose binary values are the answers to the corresponding queries. From now on, when given a query q to an oracle π the corresponding binary answer will be denoted π_q. Note that an oracle string can possibly be of exponential length (since one can specify an exponentially far location on the string using polynomially many bits).

- A PCP verifier is called *non-adaptive* if its queries are determined solely based on its input and the outcome of its coin tosses. (A general verifier, called adaptive, may determine its queries also based on answers to previously received oracle answers). From now on, whenever referring to a PCP verifier it will be assumed to be adaptive (unless otherwise specified).

- A possible motivation for the introduction of PCP systems would be to provide an alternative view of $\mathcal{NP}$, one that will rid us of the "rigidity" of the conventional view. In this regard randomness seems to be a most important ingredient, it provides us the possibility to be "imprecise" in the acceptance of false instances. This is best seen when taking the probability bound in the soundness condition to be zero. This will cause that no probability is involved in the definition and will make it collapse into $\mathcal{NP}$. To see this, notice that in the above case, the output of the verifier does not vary with the outcome of its coin tosses. This means that in order to determine the verifier's decision on some input, it suffices to examine only one of its possible executions (say, when using the all zero coin sequence). In such an execution only a polynomial portion of the PCP proof is being read by the verifier. It is easy to see, that in this case, the PCP and $\mathcal{NP}$ definitions coincide (just treat the relevant portion of the PCP proof as an $\mathcal{NP}$*-witness*).

  Note that in order to be consistent with the $\mathcal{NP}$ definition we require perfect completeness (i.e. a true instance is always accepted).

- The definition of PCP requires that for every x in L there exists a proof π_x for which it holds that $\Pr[M^{\pi_x}(x) = 1] = 1$. This means that π_x is potentially different for every x. However, we can assume w.l.o.g., that there exists a proof π which is common to all x's in L. This π will simply be the concatenation of all π_x's (according to some ordering of the x's in L). Since the verifier is polynomial we can assume that all π_x's are at most exponentially long (the verifier cannot access more than an exponentially long prefix of his proof). Therefore, the location of π_x within π will not be more than exponential in $|x|$ away, and so can be accessed in poly($|x|$) time.

- The oracle in a PCP system is viewed in a somewhat different manner than previously. We demonstrate this by comparing a PCP system to the mechanism of a *Cook-reduction*. Recall that a language L_1 is *Cook-reducible* to L_2 if there exists an oracle machine M such that for **all** $x \in \{0,1\}^*$ it holds that $M^{L_2}(x) = \chi_{L_1}(x)$. Note that the oracle in the *Cook-reduction* mechanism is the language L_2, and is supposed to exist for **all** $x \in \{0,1\}^*$ (regardless of the question whether x is in L or not). In contrast, in the case of PCP systems the oracle π is supposed **not** to exist whenever x is not in L. That is, every oracle would cause the verifer to reject x with probability at least $\frac{1}{2}$. Therefore, in the PCP case (as opposed to the *Cook-reduction* case) there is a lack of "symmetry" between the positive instances of L and the negative ones.

3 The PCP characterization of NP

3.1 Importance of Complexity Parameters in PCP Systems

As was already mentioned in subsection 2.2, the class of languages captured by PCP systems varies greatly as the appropriate parameters $r(\cdot)$ and $q(\cdot)$ are modified. This fact is demonstrated by the following assertions:

- If $\mathcal{NP} \subseteq \mathcal{PCP}(o(\log), o(\log))$ then $\mathcal{NP} = \mathcal{P}$
- $\mathcal{PCP}(\text{poly}, \text{poly}) = \mathcal{NEXP}$ $(= \mathcal{NTIME}(2^{\text{poly}}))$

By taking either one of the complexity measures to zero the definition of PCP collapses into one of the following degenerate cases:

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

When looking at the above degenerate cases of the PCP definition we do not really gain any novel view on the complexity classes involved (in this case, co$\mathcal{RP}$ and $\mathcal{NP}$). Thus, the whole point of introducing the PCP definition may be missed. What we would like to see are more delicate assertions involving both non-zero randomness and query complexity. In the following subsection we demonstrate how PCP systems can be used in order to characterize the complexity class $\mathcal{NP}$ in such a non-degenerate way. This characterization will lead to a new perspective on $\mathcal{NP}$ and enable us to further investigate the languages in it.

3.2 The PCP Theorem

As already stated, the languages in the complexity class $\mathcal{NP}$ are trivially captured by PCP systems using zero randomness and a polynomial number of queries. A natural question arises: can the two complexity measures be traded off, in a way that still captures the class $\mathcal{NP}$? Most surprisingly, not only the answer to the above question is positive, but also a most powerful result emerges. The number of queries made by the verifier can be brought down to a constant while using only a logarithmic number of coin tosses. This result is known as the PCP theorem (it will be cited here without a proof).

Our goal is to characterize $\mathcal{NP}$ in terms of PCP systems. We start by demonstrating how $\mathcal{NP}$ upper bounds a fairly large class in the PCP hierarchy. This is the class of languages having a PCP system whose verifier makes a polynomial number of queries while using a logarithmic number of coin tosses.

Proposition 3.1 $\mathcal{PCP}(\log, \text{poly}) \subseteq \mathcal{NP}$

Proof: Let L be a language in $PCP(\log, \text{poly})$. We will show how to use its PCP system in order to construct a non-deterministic machine M which decides L in polynomial-time. This will imply that L is in $\mathcal{NP}$.

Let M' be the probabilistic-polynomial time oracle machine in the above $\mathcal{PCP}(\log, \text{poly})$ system for L. We are guaranteed that on input $x \in \{0,1\}^*$, M' makes poly$(|x|)$ queries using $O(\log(|x|))$ coin tosses. For the sake of simplicity, we prove the claim for a non-adaptive M' (in order to adjust the proof to the adaptive case, some minor modifications are required).

Denote by $\langle r_1, \ldots, r_m \rangle$ the sequence of all m possible outcomes of the coin tosses made by M' (note that $|r_i| = O(\log(|x|))$ and $m = 2^{O(\log(|x|))} = \text{poly}(|x|)$). Denote by $\langle q_1^i, \ldots, q_{n_i}^i \rangle$ the sequence of n_i queries made by M when using the coin sequence r_i (note that n_i is potentially different for each i, and is polynomial in $|x|$). Since M' is non-adaptive, its queries are determined as a function of the input x and the coin sequence r_i, and do not depend on answers to previous queries.

By the completeness condition we are guaranteed that for every x in L there exists a PCP proof π_x, such that the verifier M' always accepts x when given access to π_x. A natural candidate for an $\mathcal{NP}$*-witness* for x would be π_x. However, as already stated in subsection 2.3, π_x might be of exponential size in $|x|$, and therefore unsuitable to be used as an $\mathcal{NP}$*-witness*. We will therefore use a "compressed" version of π_x, this version corresponds to the portion of the proof which is actually being read by the verifier M'.

We now turn to the construction of a witness w, given $x \in L$ and a corresponding oracle π_x (for the sake of simplicity we denote it by π). Consider all possible executions of M' on input x given access to the oracle string π (each execution depends on the coin sequence r_i). Take the substring of π containing all the bits examined by M' during these executions (i.e. $\left\{ \langle \pi_{q_1^i}, \ldots, \pi_{q_{n_i}^i} \rangle \right\}_{i=1}^m$). Encode each entry in this substring as $\langle index, \pi_{index} \rangle$ (that is, $\langle query, answer \rangle$), denote the resulting encoded string by w_x^π (note that now $|w_x^\pi|$ is polynomial in $|x|$).

We now describe the non-deterministic machine M which decides L in polynomial time. Given input x, and w on the guess tape, M will simulate the execution of M' on input x for all possible r_i's. Every query made by M' will be answered by M according to the corresponding answers appearing in w (by performing binary search on the indices in w). The machine M will accept *if and only if* M' would have accepted x for **all** possible r_i's.

Since M simulates the execution of M' exactly m times (which is polynomial in $|x|$), and since M' is a polynomial time machine, then M is itself a polynomial-time machine, as required. It remains to be seen that $L(M)$ indeed equals L:

- $\forall x \in L$, we show that there exists w such that $M(x, w) = 1$. By the perfect completeness condition of the PCP system for L, there exists an oracle π such that $\Pr[M'^{\pi}(x) = 1] = 1$. Therefore, it holds that for **all** coin sequences r_i, the machine M' accepts x while accessing π. It immediately follows by definition that $M(x, w_x^\pi) = 1$, where w_x^π is as described above.

- $\forall x \notin L$, we show that for **all** w's it holds that $M(x, w) = 0$. By the soundness condition of the PCP system for L, for all oracles π it holds that $\Pr[M'^{\pi}(x) = 1] \leq \frac{1}{2}$. Therefore, for at least $\frac{1}{2}$ of the possible coin sequences r_i, M does not accept x while accessing π. Assume, for the sake of contradiction, that there exists a witness w for which it holds that $M(x, w) = 1$. By the definition of M this means that for all possible coin tosses M' accepts x when given answers from w. We can therefore use w in order to construct an oracle, π^w, for which it holds that $\Pr\left[M'^{\pi^w}(x) = 1\right] = 1$, in contradiction to the soundness condition. (the oracle π^w can be constructed as follows: for every index q that appears in w, define π_q^w to be the binary answer corresponding to q. Define the rest of π^w arbitrarily.)

Consider now the case of an adaptive M'. In this case, we can construct w_x^π adaptively. Given an input $x \in L$ and a corresponding oracle π, run M'^{π} on x for every random string r_i, and see what are the queries made by M' (which depend on x, r_i and answers to previous queries). Then take w_x^π to be the substring of π that is defined by all these queries, as before. ■

The essence of the above proof, is that given a PCP proof (of logarithmic randomness) for some x in L we can efficiently "pack" it (compress it into polynomial size) and transform it into an $\mathcal{NP}$-witness for x. This is due to the fact that the **total** portion of the proof used by the verifier (in all possible runs, i.e. over all possible coin sequences) is bounded by a polynomial. In light of the above, any result of the type

$$\mathcal{NP} \subseteq \mathcal{PCP}(\log, q(\cdot))$$

would be interesting, since it implies that for every $x \in L$, we can construct a witness with the additional property, that enables a "lazy" verifier to toss coins, and decide membership in L, based only on a tiny portion of the $\mathcal{NP}$-witness (as will be further discussed in subsection 3.3).

It turns out that the polynomial $q(\cdot)$ bounding the number of queries in a result of the above kind can be taken to be a constant. This surprising result is what we refer to as the PCP theorem.

Theorem 3 (The PCP Theorem)

$$\mathcal{NP} \subseteq \mathcal{PCP}(\log, O(1))$$

The PCP theorem is a culmination of a sequence of works, each establishing a meaningful and increasingly stronger statement. The proof of the PCP theorem is one of the most complicated proofs in the theory of computation and it is beyond our scope to prove it here. We state as a side remark, that the smallest possible number of queries for which the PCP theorem has been proven is currently 5 (whereas with 3 queries one can get arbitrarily close to soundness error 1/2).

The conclusion is that $\mathcal{NP}$ is *exactly* the set of languages which have a PCP verifier that asks a constant number of queries using a logarithmic number of coin tosses.

Corollary 4 (The PCP Characterization of $\mathcal{NP}$)

$$\mathcal{NP} = \mathcal{PCP}(\log, O(1))$$

Proof: Combining Theorem 3 with Proposition 3.1, we obtain the desired result. ■

3.3 The PCP Theorem gives rise to "robust" $\mathcal{NP}$-relations

Recall that every language L in $\mathcal{NP}$ can be associated with an $\mathcal{NP}$-relation R_L (in case the language is natural, so is the relation). This relation consists of all pairs (x, y) where x is a positive instance of L and y is a corresponding *$\mathcal{NP}$-witness*. The PCP theorem gives rise to another (unnatural) relation R'_L with some extra properties. In the following subsection we briefly discuss some of the issues regarding the relation R'_L.

Since every $L \in \mathcal{NP}$ has a $\mathcal{PCP}(\log, O(1))$ system we are guaranteed that for every x in L there exists a PCP proof π_x, such that the corresponding verifier machine M always accepts x when given access to π_x. In order to define our relation we would like to consider pairs of the form (x, π_x). However, in general, π_x might be of exponential size in $|x|$, and therefore unsuitable to be used in an $\mathcal{NP}$-relation. In order to "compress" it into polynomial size we can use the construction introduced in the proof of Proposition 3.1 (i.e. of a witness w for the non-deterministic machine M). Denote by π'_x the resulting "compressed" version of π_x. We are now ready to define the relation:

$$R'_L \stackrel{\text{def}}{=} \{(x, \pi'_x) \mid \Pr\left[M^{\pi_x}(x) = 1\right] = 1\}$$

By the definition of PCP it is obvious that $x \in L$ *if and only if* there exists π'_x such that $(x, \pi'_x) \in R'_L$. It follows from the details in the proof of proposition 3.1 that R'_L is indeed recognizable in polynomial-time.

Although not stated in the theorem, the proof of the PCP theorem actually demonstrates how to efficiently transform an $\mathcal{NP}$-witness y (for an instance x of $L \in \mathcal{NP}$) into an oracle proof $\pi_{x,y}$

for which the PCP verifier always accepts x. Thus, there is a Levin-reduction between the natural $\mathcal{NP}$-relation for L and R'_L.

We conclude that any $\mathcal{NP}$-witness of R_L can be efficiently transformed into an $\mathcal{NP}$-witness of R'_L (i.e. an oracle proof) which offers a trade-off between the portion of the $\mathcal{NP}$-witness being read by the verifier and the amount of certainty it has in its answer. That is, if the verifier is willing to tolerate an error probability of 2^{-k}, it needs to inspect $O(k)$ bits of the proof (the verifier chooses k random strings $r_1, \ldots, r_k$ uniformly among $\{0,1\}^{O(\log)}$. It will be convinced with probability 2^{-k} that the input x is in L, if for every i, M accepts x using randomness r_i and given oracle access to the appropriate $O(1)$ queries).

3.4 Simplifying assumptions about $\mathcal{PCP}(\log, O(1))$ verifiers

When considering a $\mathcal{PCP}(\log, O(1))$ system, some simplifying assumptions about the corresponding verifier machine can be made. We now turn to introduce two of them:

1. Any verifier in a $\mathcal{PCP}(\log, O(1))$ system can be assumed to be non-adaptive (i.e. its queries are determined as a function of the input and the random tape only, and do not depend on answers to previous queries). This is due to the fact that any adaptive $\mathcal{PCP}(\log, O(1))$ verifier can be converted into a non-adaptive one by modifying it in such a way that it will consider **all** possible sequences of $\{0,1\}$ answers given to its queries by the oracle. This certainly costs us in an exponential blowup in the query complexity, but, since the number of queries made by the original (adaptive) verifier is constant, so will be the query complexity of the modified (non-adaptive) verifier after the blowup. Note that in general, adaptive verifiers *are* more powerful than non-adaptive ones (in terms of quantitative results). There are constructions in which adaptive verifiers make less queries than non-adaptive ones while achieving the same results.

2. Any verifier in a $\mathcal{PCP}(\log, O(1))$ system can be assumed to always make the same (constant) number of queries (regardless of the outcome of its coin tosses). Take any verifier in a $\mathcal{PCP}(\log, O(1))$ system not satisfying the above property. Let t be the maximal number of queries made in some execution of the above verifier (over all possible outcomes of the coin tosses). For every possible outcome of the coin tosses, modify the verifier in such a way that it will ask a total number of t queries, make him ignore answers to the newly added queries. Clearly, such a verifier will be consistent with the original one, and will still make only a constant number of queries (which is t).

From now on, whenever referring to $\mathcal{PCP}(\log, O(1))$ systems, free use of the above assumptions will be made (without any loss of generality).

4 PCP and non-approximability

Many natural optimization problems are known to be $\mathcal{NP}$-*hard*. However, many times an approximation to the exact value of the solution could be sufficient for our needs. In this section we will investigate the existence (or rather, the inexistence) of efficient approximation algorithms for two $\mathcal{NP}$-*complete* problems, namely, $max3SAT$ and $maxCLIQUE$.

An algorithm for a given problem is considered a C-*approximation* algorithm if for every instance it generates an answer that is off the correct answer by a factor of at most C. The question of interest, is given an $\mathcal{NP}$-*complete* problem Π, what is the best C for which there is a C-approximation algorithm for Π.

The PCP characterization of $\mathcal{NP}$ provides us an alternative view of languages in $\mathcal{NP}$. This view is not as rigid as the original one, and thus creates a framework which is apparently more insightful for the study of approximability.

We start by rephrasing the PCP theorem in an alternative way. This in turn will be used in order to derive an immediate non-approximability result for $max3SAT$. While rephrasing the PCP theorem, a new type of polynomial-time reductions, which we call *amplifying*, emerges.

4.1 Amplifying Reductions

Consider an unsatisfiable $3CNF$ formula[1]. It may be the case that the formula is very "close" to being satisfiable. For example, there exist unsatisfiable formulae such that by removing only one of their clauses, they suddenly become satisfiable.

In contrast, there exist unsatisfiable $3CNF$ formulae which are much "farther" from being satisfiable than the above mentioned formulae. These formulae may always have a constant fraction of unsatisfied clauses (for all possible truth assignments). As a consequence, they offer us the (most attractive) feature of being able to probabilistically check whether a certain truth assignment satisfies them or not (by randomly sampling their clauses and picking with constant probability a clause which is unsatisfied by this assignment). Not surprisingly, this resembles the features of a PCP system.

Loosely speaking, amplifying reductions of 3SAT [2] to itself are *Karp-reductions*, which, in addition to the conventional properties, have the property that they map unsatisfiable $3CNF$ formulae into unsatisfiable $3CNF$ formulae which are "far" from being satisfiable (in the above sense).

Definition 5 (amplifying reduction) *An* amplifying reduction *of* 3SAT *to itself is a polynomial-time computable function f mapping the set of* $3CNF$ *formulae to itself such that for some constant* $\epsilon > 0$ *it holds that:*

- *f maps satisfiable* $3CNF$ *formulae to satisfiable* $3CNF$ *formulae.*
- *f maps non-satisfiable* $3CNF$ *formulae to (non-satisfiable)* $3CNF$ *formulae for which every truth assignment satisfies at most an* $1 - \epsilon$ *fraction of the clauses.*

An amplifying reduction of a language L in $\mathcal{NP}$ to 3SAT, can be defined analogously.

4.2 PCP Theorem Rephrased

Amplifying reductions seem like a suitable tool to be used in order to construct a PCP system for every language in $\mathcal{NP}$. Not only they are efficiently computable, but they enable us to map negative instances of any language in $\mathcal{NP}$ into negative instances of 3SAT which we may be able to reject on a probabilistic basis (analogously to the soundness condition in the PCP definition).

It turns out that the converse is also true, given a PCP system for a language in $\mathcal{NP}$ we are also able to construct an amplifying reduction of 3SAT to itself.

Theorem 6 (PCP theorem rephrased) *The following are equivalent:*

1. $\mathcal{NP} \subseteq \mathcal{PCP}(\log, O(1))$. (The PCP Theorem).

[1] Recall that a t-CNF formula is a boolean formula consisting of a conjunction of clauses, where each clause is a disjunction of up to t literals (a literal is a variable or its negation).

[2] 3SAT is the problem of deciding whether a given $3CNF$ formula has a satisfying truth assignment.

2. *There exists an amplifying reduction of* 3SAT *to itself.*

Proof: We start with the ((1) $\Rightarrow$ (2)) direction. Consider any language $L \in \mathcal{NP}$. By the PCP theorem L has a $\mathcal{PCP}(\log, O(1))$ system, we will now show how to use this system in order to construct an amplifying reduction from L to 3SAT. This will in particular hold for L = 3SAT (which is itself in $\mathcal{NP}$), and the claim will follow.

Let M be the probabilistic polynomial-time oracle machine in the above $\mathcal{PCP}(\log, O(1))$ system for L. We are guaranteed that on input $x \in \{0,1\}^*$, M makes $t = O(1)$ queries using $O(\log(|x|))$ coin tosses.

Denote by $\langle r_1, \ldots, r_m \rangle$ the sequence of all m possible outcomes of the coin tosses made by M (note that $|r_i| = O(\log(|x|))$ and $m = 2^{O(\log(|x|))} = \text{poly}(|x|)$).

Denote by $\langle q_1^i, \ldots, q_t^i \rangle$ the sequence of t queries made by M when using the coin sequence r_i. As mentioned in subsection 3.4, we can assume that M is non-adaptive, therefore its queries are determined as a function of the input x and the coin sequence r_i, and do not depend on answers to previous queries (although not evident from the notation q_j^i, the queries do not depend only on r_i, but on x as well).

We now turn to the construction of the amplifying reduction. Given $x \in \{0,1\}^*$, we construct for each r_i a (constant size) $3CNF$ boolean formula, φ_i^x, describing whether M would have accepted the input x (i.e. describing all possible outputs of M on input x, using the coin sequence r_i). We associate to each query q_j^i a boolean variable $z_{q_j^i}$ whose value should be the answer M gets to the corresponding query. Again, since M is assumed to be non-adaptive, when given its input and coin tosses, M's decision is completely determined by the answers it gets to its queries. In other words, M's decision depends only on the values of $\langle z_{q_1^i}, \ldots, z_{q_t^i} \rangle$.

In order to construct φ_i^x, begin by computing the following truth table: to every possible sequence $\langle z_{q_1^i}, \ldots, z_{q_t^i} \rangle$ assign the corresponding boolean decision of M (i.e. the output of M on input x, using the coin sequence r_i, and given answers $z_{q_j^i}$ to queries q_j^i). Clearly, this can be computed in polynomial-time (by simulating M's execution). Therefore, the whole table can be computed in polynomial-time (since the number of possible assignments to $\langle z_{q_1^i}, \ldots, z_{q_t^i} \rangle$ is 2^t, which is a constant). We can now build a $3CNF$ boolean formula, φ_i^x, which is consistent with the above truth table, this is done in the following way:

1. Construct a *t-CNF* formula $\psi_i^x = \psi_i^x(z_{q_1^i}, \ldots, z_{q_t^i})$ which is consistent with the truth table.

2. Using a constant number of auxiliary variables, transform it to $3CNF$ (denoted φ_i^x).

Since the table size is constant, the above procedure can be executed in constant time. Note that in the transformation of *t-CNF* formulae into $3CNF$ formuae, each clause with t literals is substituted by at most t clauses of 3 literals. Since ψ_i^x consists of exactly 2^t clauses we conclude that the number of clauses in φ_i^x is bounded by $t \cdot 2^t$.

Finally, given φ_i^x for $i = 1, \ldots, m$, we let our amplifying reduction f map $x \in \{0,1\}^*$ into the $3CNF$ boolean formula:

$$\varphi^x \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m} \varphi_i^x$$

Since for every $i = 1, \ldots, m$ the (constant size) formula φ_i^x can be computed in polynomial-time (in $|x|$), and since $m = \text{poly}(|x|)$, it follows that the mapping $f : x \mapsto \varphi^x$ is polynomial-time computable, and $|\varphi^x|$ is polynomial in $|x|$ (note also that the number of clauses in φ^x is bounded by $m \cdot t \cdot 2^t$). It remains to be verified that f is indeed an amplifying reduction:

- $\forall x \in L$, we now show that φ^x is in 3SAT, this happens if and only if the corresponding t-CNF formula $\psi^x \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m} \psi_i^x(z_{q_1^i}, \ldots, z_{q_t^i})$ is in t-SAT (recall that ψ_i^x was introduced in the construction of φ_i^x). Since $L \in$ PCP, then there exists an oracle π such that $\Pr[M^\pi(x) = 1] = 1$. Therefore, it holds that for every coin sequence r_i, the machine M accepts x while accessing π. Since ψ_i^x is consistent with the above mentioned truth table it follows that for **all** $i = 1, \ldots, m$, it holds that $\psi_i^x(\pi_{q_1^i}, \ldots, \pi_{q_t^i}) = 1$, and thus ψ^x is in t-SAT. We conclude that φ^x is in 3SAT, as required [3].

- $\forall x \notin L$, we now show that every truth assignment satisfies at most an $1 - \epsilon$ fraction of φ^x's clauses. Since $L \in$ PCP, then for all oracles π it holds that $\Pr[M^\pi(x) = 1] \leq \frac{1}{2}$. Therefore, for at least $\frac{1}{2}$ of the possible coin sequences r_i, machine M does not accept x while accessing π. Put in other words, for each truth assignment (which corresponds to some π) at least $\frac{1}{2}$ of the φ_i^x's are unsatisfiable. Since every unsatisfiable boolean formula always has at least one unsatisfied clause, it follows that for every truth assignment φ^x has at least $\frac{m}{2}$ unsatisfied clauses. Since the number of clauses in φ^x is bounded by $m \cdot t \cdot 2^t$, by taking ϵ to be the constant $\frac{1}{2 \cdot t \cdot 2^t}$ we are guaranteed that every truth assignment satisfies at most an $1-\epsilon$ fraction of φ^x's clauses.

We now turn to the ((2) $\Rightarrow$ (1)) direction. Under the assumption that there exists an amplifying reduction of 3SAT to itself we will show that the PCP theorem holds. Consider any language $L \in \mathcal{NP}$. Since L is *Karp-reducible* to 3SAT, it is sufficient to show that 3SAT $\in \mathcal{PCP}(\log, O(1))$.

Let $f : 3CNF \rightarrow 3CNF$ be an amplifying reduction of 3SAT to itself. And let ϵ be the constant guaranteed by Definition 5. We now show how to use f in order to construct a $\mathcal{PCP}(\log, O(1))$ system for 3SAT. We start by giving an informal description of the verifier machine M. Given a certain $3CNF$ formula φ, M computes $\varphi' = f(\varphi)$. It then tosses coins in order to uniformly choose one of the clauses of φ'. By querying the oracle string (which should be a possible truth assignment for φ') M will assign truth values to the chosen clause's variables. M will accept if and only if the clause is satisfied. The fact that f is an amplifying reduction implies that whenever M gets a negative instance of 3SAT, with constant probability the chosen clause will not be satisfied. In contrast, this will never happen when looking at a positive instance.

We now turn to a more formal definition of the PCP verifier machine M. On input $\varphi \in 3CNF$ and given access to an oracle string π', M is defined in the following way:

1. Find the $3CNF$ formula $\varphi' = \varphi'(x_1, \ldots, x_{n'}) \stackrel{\text{def}}{=} f(\varphi)$.

   $\varphi' = \bigwedge_{i=1}^{m'} c_i$ where c_i denotes a clause with 3 literals.

2. Select a clause c_i of φ' uniformly.

   Denote by $\langle x_{i_1}, x_{i_2}, x_{i_3} \rangle$ the three variables whose literals appear in c_i.

3. Query the values of $\langle \pi'_{i_1}, \pi'_{i_2}, \pi'_{i_3} \rangle$ separately, and assign them to $\langle x_{i_1}, x_{i_2}, x_{i_3} \rangle$ accordingly. Verify the truth value of $c_i = c_i(x_{i_1}, x_{i_2}, x_{i_3})$.

4. Repeat stages 2,3 for $\lceil \frac{1}{\epsilon} \rceil$ times independently (note that $\lceil \frac{1}{\epsilon} \rceil$ is constant).

5. Output 1 if and only if in **all** iterations the truth value of c_i was 1.

[3] Note that all φ_i^x's have disjoint sets of auxiliary variables, hence transforming a satisfying assignment of ψ^x into a satisfying assignment of φ^x causes no inconsistencies.

Clearly, M is a polynomial-time machine. Note that f is computable in polynomial-time (this also implies that $n', m' = \text{poly}(|\varphi|)$). In addition, the number of iterations executed by M is constant, and in each iteration a polynomial amount of work is executed (depending on n', m' which are, as already mentioned, polynomial in $|\varphi|$).

We turn to evaluate the additional complexity measures involved. In terms of randomness, M needs to uniformly choose $\lceil \frac{1}{\epsilon} \rceil$ numbers in the set $\{1, ..., m'\}$. This involves $O(\log(m')) = O(\log(|\varphi|))$ coin tosses, as required. In terms of queries, the number of queries asked by M is exactly $\frac{3}{\epsilon}$ which is constant, again as required. It remains to examine the completeness and soundness of the above PCP system:

- *completeness*: If $\varphi \in$ 3SAT, then $\varphi' \in$ 3SAT (since f is an amplifying reduction). Therefore there exists a truth assignment, π', such that $\varphi'(\pi') = 1$. Now, since every clause of φ' is satisfied by π', it immediately follows that:
$$\Pr\left[M^{\pi'}(\varphi) = 1\right] = 1$$
- *soundness*: If $\varphi \notin$ 3SAT then any truth assignment for φ' satisfies at most an $1 - \epsilon$ fraction of the clauses. Therefore for any possible truth assignment (oracle) π' it holds that
$$\Pr\left[M^{\pi'}(\varphi) = 1\right] = \Pr\left[\bigwedge_{j=1}^{\lceil \frac{1}{\epsilon} \rceil} \left\{ \begin{array}{c} c_{i_j} \text{ is satisfied by} \\ \text{the assignment } \pi' \end{array} \right\}\right] \leq (1-\epsilon)^{\lceil \frac{1}{\epsilon} \rceil} \leq \frac{1}{e} < \frac{1}{2}$$

where the probability is taken over M's internal coin tosses (i.e. over the choice of $i_1, \ldots, i_{\lceil \frac{1}{\epsilon} \rceil}$). ■

Corollary 7 *There exists an amplifying reduction of* 3SAT *to itself.*

Proof: Combining the PCP Theorem with Theorem 6, we obtain the desired result. ■

4.3 Connecting PCP and non-approximability

The characterization of $\mathcal{NP}$ using probabilistic checkable proof systems enabled the area of approximability to make a significant progress.

In general, PCP systems for $\mathcal{NP}$ yield strong non-approximability results for various classical optimization problems. The hardness of approximation is typically established using the notion of gap problems, which are a particular case of promise problems. Recall that a promise problem consists of two sets (A, B), where A is the set of YES instances and B is the set of NO instances. A and B need not be complementary, that is, an instance $x \in \{0,1\}^*$ is not necessarily in either A or B. We demonstrate the notion of a gap problem using the promise problem $gapCLIQUE$ as an example.

Denote by $maxCLIQUE(G)$ the size of the maximal clique in a graph G. Let $gapCLIQUE_{\alpha,\beta}$ be the promise problem (A, B) where A is the set of all graphs G with $maxCLIQUE(G) \geq \alpha$, and B is the set of all graphs G with $maxCLIQUE(G) \leq \beta$. The gap is defined as α/β. Typically, a hardness result will specify a value C of the gap for which the problem is $\mathcal{NP}$*-hard*. This means that there is no efficient algorithm that approximates the $maxCLIQUE$ size of a graph G within a factor of C (unless $\mathcal{NP} = \mathcal{P}$).

The gap versions of various other optimization problems are defined in an analogous way. In this subsection we bring two non-approximability results concerning the problems $max3SAT$ and $maxCLIQUE$ that will be defined in the sequel.

4.3.1 An immediate non-approximability result for max3SAT

Definition 8 *(max3SAT): Define max3SAT to be the following problem: Given a 3CNF boolean formula φ find the maximal number of clauses that can be simultaneously satisfied by any truth assignment to the variables of φ.*

$max3SAT$ is known to be $\mathcal{NP}$*-hard*. Therefore, approximating it would be desirable. This motivates the definition of the corresponding gap problem:

Definition 9 $(gap3SAT_{\alpha,\beta})$ *: Let $\alpha, \beta \in [0,1]$ such that, $\alpha \geq \beta$.*
Define $gap3SAT_{\alpha,\beta}$ to be the following promise problem:

- *The YES instances are all $3CNF$ formulae φ, such that there exists a truth assignment which satisfies at least an α-fraction of the clauses of φ.*
- *The NO instances are all $3CNF$ formulae φ, such that every truth assignment satisfies less than a β-fraction of the clauses of φ.*

Note that $gap3SAT_{1,1}$ is an alternative formulation of 3SAT (the decision problem).

The following claim states that, for some $\beta < 1$, it is $\mathcal{NP}$*-hard* to distinguish between satisfiable $3CNF$ formulae and $3CNF$ formulae for which no truth assignment satisfies more than a β-fraction of its clauses. This result implies that there is some constant $C > 1$ such that $max3SAT$ could not be approximated within C (unless $\mathcal{NP} = \mathcal{P}$). The claim is an immediate result of Corollary 7.

Claim 4.1 *There exists a constant $\beta < 1$, such that the promise problem $gap3SAT_{1,\beta}$ is $\mathcal{NP}$-hard.*

Proof: Let $L \in \mathcal{NP}$. We want to manifest that L is *Karp-reducible* to $gap3SAT_{1,\beta}$.

$3SAT$ is $\mathcal{NP}$-complete, therefore there exists a *Karp-reduction* f_1 from L to $3SAT$. By Corollary 7 there exists an amplifying reduction f_2 (and a constant $\epsilon > 0$) from $3SAT$ to itself. Now, take any $1 - \epsilon < \beta < 1$:

- For $x \in L$, $\varphi = f_2(f_1(x))$ is satisfiable, and is therefore a YES instances of $gap3SAT_{1,\beta}$.
- For $x \notin L$, $\varphi = f_2(f_1(x))$ is not satisfiable. Furthermore, for every truth assignment, the fraction of satisfied clauses in φ is at most $1-\epsilon$. Therefore, φ is a NO instance of $gap3SAT_{1,\beta}$.

■

Recently, stronger results were proven. These results show that for every $\beta > 7/8$, the problem $gap3SAT_{1,\beta}$ is $\mathcal{NP}$*-hard*. This means that it is infeasible to come with an efficient algorithm that approximates $max3SAT$ within a factor strictly smaller than $8/7$. On the other hand, $gap3SAT_{1,7/8}$ is known to be polynomially solvable, and therefore the $8/7$-approximation ratio is tight.

4.3.2 MaxCLIQUE is non-approximable within a factor of two

We briefly review the definitions of the problems $maxCLIQUE$ and $gapCLIQUE_{\alpha,\beta}$:

Definition 10 *(maxCLIQUE): Define maxCLIQUE to be the following problem: Given a graph G, find the size of the maximal clique of G (a clique is a set of vertices such that every pair of vertices share an edge).*

$maxCLIQUE$ is known to be $\mathcal{NP}$*-hard*. Therefore, approximating it would be desirable. This motivates the definition of the corresponding gap problem:

Definition 11 *($gapCLIQUE_{\alpha,\beta}$) : Let $\alpha, \beta : N \to N$ be two functions, satisfying $\alpha(n) \geq \beta(n)$ for every n. For a graph G, denote $|G|$ to be the number of vertices in G.*

Define $gapCLIQUE_{\alpha,\beta}$ to be the following promise problem:

- *The YES instances are all the graphs G, with max clique greater than or equal to $\alpha(|G|)$.*
- *The NO instances are all the graphs G, with max clique smaller than or equal to $\beta(|G|)$.*

We conclude our discussion on PCP systems by presenting a nice theorem which demonstrates the hardness of approximating $maxCLIQUE$. The theorem implies that it is an infeasible task to approximate $maxCLIQUE$ within a constant smaller than two (unless $\mathcal{NP} = \mathcal{P}$).

Note, however, that this is not the strongest result known. It has been shown recently that given a graph G of size N, the value of $maxCLIQUE$ is non-approximable within a factor of $N^{1-\epsilon}$ (for every $\epsilon > 0$). This result is tight, since an $N^{1-o(1)}$-approximation algorithm is known to exist (the latter is scarcely better than the trivial approximation factor of N).

Theorem 12 *There exists a function $\alpha : N \to N$, such that the promise problem $gapCLIQUE_{\alpha,\alpha/2}$ is $\mathcal{NP}$-hard.*

Proof: Let $L \in \mathcal{NP}$ be some language. We want to show that L is *Karp-reducible* to the language $gapCLIQUE_{\alpha,\alpha/2}$ (for some function $\alpha : N \to N$ which is not dependent on L, rather it is common to all L's).

Loosely speaking, given input $x \in \{0,1\}^*$ we construct in an efficient way a graph G_x having the following property: If x is in L then G_x is a YES instance of *gapCLIQUE*, whereas, if x is not in L then G_x is a NO instance of *gapCLIQUE*.

We now turn to a formal definition of the above mentioned reduction. By the PCP-theorem, L has a $PCP(O(\log), O(1))$ system. Therefore, there exists a probabilistic polynomial-time oracle machine M, that on input $x \in \{0,1\}^*$ makes $t = O(1)$ queries using $O(\log(|x|))$ random coin tosses.

Again, we let $\langle r_1, \ldots, r_m \rangle$ be the sequence of all m possible outcomes of the coin tosses made by M (note that $m = \text{poly}(|x|)$).

Let $\langle q_1^i, \ldots, q_t^i \rangle$ denote the t queries made by M when using the coin tosses r_i, and let $\langle a_1^i, \ldots, a_t^i \rangle$ be a possible sequence of answers to the corresponding queries. We now turn to define a graph G'_x that corresponds to machine M and input x:

vertices: For every possible r_i, the tuple

$$(\mathbf{r_i}, (\mathbf{q_1^i}, a_1^i), \ldots, (\mathbf{q_t^i}, a_t^i))$$

is a vertex in G'_x *if and only if* when using r_i, and given answers a_j^i to queries q_j^i, M accepts x. Note that since M is non-adaptive, once r_i is fixed then so are the queries $\langle q_1^i, \ldots, q_t^i \rangle$. This implies that two vertices having the same r_i, also have the same q_j^i's. Therefore, the number of vertices corresponding to a certain r_i is smaller or equal to 2^t, and the total number of vertices in G'_x is smaller or equal to $m \cdot 2^t$.

edges: Two vertices $v = (\mathbf{r_i}, (\mathbf{q_1^i}, a_1^i), \ldots, (\mathbf{q_t^i}, a_t^i))$ and $u = (\mathbf{r_j}, (\mathbf{q_1^j}, a_1^j), \ldots, (\mathbf{q_t^j}, a_t^j))$ will **not** have an edge between them *if and only if* they are not *consistent*, that is, v and u contain the same query and each one of them has a different answer to this query.

Note that if u and v contain the same randomness (i.e. r_i is equal to r_j) they do not share an edge, since they cannot be consistent (as mentioned earlier, vertices having the same randomness have also the same queries, so u and v must differ in the answers to the queries).

Finally, modify G'_x by adding to it $(m \cdot 2^t - |G'_x|)$ isolated vertices. The resulting graph will have exactly $m \cdot 2^t$ vertices and will be denoted G_x. Note that since the above modification does not add any edges to G'_x, it does not change the size of any clique in G'_x (in particular it holds that $maxCLIQUE(G_x) = maxCLIQUE(G'_x)$).

The above reduction is efficient, since the graph G_x can be constructed in polynomial time: There are at most $m \cdot 2^t$ vertices in G'_x(which is polynomial in $|x|$ since t is a constant), and to decide whether $(\mathbf{r_i}, (\mathbf{q_1^i}, a_1^i), \ldots, (\mathbf{q_t^i}, a_t^i))$ is a vertex in G'_x, one has to simulate machine M on input x, randomness r_i, queries $\{q_j^i\}_{j=1}^t$ and answers $\{a_j^i\}_{j=1}^t$ and see whether it accepts or not. This is, of course, polynomial, since M is polynomial. Finally, deciding whether two vertices share an edge can be done in polynomial time.

Let $\alpha(n) \stackrel{\text{def}}{=} n/2^t$. Since $|G_x| = m \cdot 2^t$, it holds that $\alpha(|G_x|) = m$. It is therefore sufficient to show a reduction from the language L to $gapCLIQUE_{m,m/2}$, this will imply that the promise problem $gapCLIQUE_{\alpha,\alpha/2}$ is $\mathcal{NP}$-*hard*.

- For $x \in L$, we show that G_x contains a clique of size m. By the PCP definition there exists a proof π such that for every random string r, machine M accepts x using randomness r and given oracle access to π. Look at the following set of m vertices in the graph G_x: $S = \{(\mathbf{r_i}, (\mathbf{q_1^i}, \pi_{q_1^i}), \ldots, (\mathbf{q_t^i}, \pi_{q_t^i}))$ for $1 \le i \le m\}$. It is easy to see that all the vertices in S are indeed legal vertices, because π is a proof for x. Also, all the vertices in S must be consistent, because all their answers are given according to π, and therefore, every two vertices in S share an edge. This entails that S is an m-clique in G_x, and therefore G_x is a YES instance of $gapCLIQUE_{m,m/2}$.

- For $x \notin L$, we show that G_x does not contain a clique of size greater than $m/2$. Suppose, in contradiction, that S is a clique in G_x of size greater than $m/2$. Define now the following proof π: For every query and answer (q, a) in one of the vertices of S define $\pi_q = a$. For every other query (which is not included in either of the vertices of S) define π_q to be an arbitrary value in {0,1}. Since S is a clique, all its vertices share an edge and are therefore consistent. Note that π is well defined, the consistency requirement implies that same queries have same answers (for all queries and answers appearing in some vertex in S). Therefore, it cannot be the case that we give two incosistent values to the same entry in π during its construction. Now, since all the vertices of S have different r_i's and $|S|$ is greater than $m/2$, it holds that for more than $\frac{1}{2}$ of the possible coin sequences r_i, machine M accepts x while accessing π. In other words, $\Pr[M^\pi(x) = 1] > 1/2$, in contradiction to the soundness condition. We conclude that indeed G_x does not have a clique of size greater than $m/2$, and is therefore a NO instance of $gapCLIQUE_{m,m/2}$.

■