Computational Complexity: A Modern Approach

Sanjeev Arora and Boaz Barak
Princeton University

http://www.cs.princeton.edu/theory/complexity/

complexitybook@gmail.com

Chapter 11

PCP Theorem and Hardness of Approximation: An introduction

> *"...most problem reductions do not create or preserve such gaps...To create such a gap in the generic reduction (cf. Cook)...also seems doubtful. The intuitive reason is that computation is an inherently unstable, non-robust mathematical object, in the the sense that it can be turned from non-accepting to accepting by changes that would be insignificant in any reasonable metric."*
> Papadimitriou and Yannakakis [PY88]

This chapter describes the **PCP** Theorem, a surprising discovery of complexity theory, with many implications to algorithm design. Since the discovery of **NP**-completeness in 1972 researchers had mulled over the issue of whether we can efficiently compute *approximate* solutions to **NP**-hard optimization problems. They failed to design such approximation algorithms for most problems (see Section 1.1 for an introduction to approximation algorithms). They then tried to show that computing approximate solutions is also hard, but apart from a few isolated successes this effort also stalled. Researchers slowly began to realize that the Cook-Levin-Karp style reductions do not suffice to prove any limits on approximation algorithms (see the above quote from an influential Papadimitriou-Yannakakis paper that appeared a few years before the discoveries described in this chapter). The **PCP** theorem, discovered in 1992, gave a new definition of **NP** and provided a new starting point for reductions. It was considered very surprising at the time (see the note at the end of Section 1.2.2).

As we discuss in Section 1.2, there are two ways to view the **PCP** theorem. One view of the **PCP** Theorem is that it constructs *locally testable proof systems*: the **PCP** Theorem gives a way to transform every mathematical proof into a form that is checkable by only looking at very few (probabilistically chosen) symbols of the proof. (The acronym "PCP" stands for *Probabilistically Checkable Proofs*.) Another view of the **PCP** Theorem is that it is a *hardness of approximation* result: the **PCP** theorem shows that for many **NP**-complete optimization problems, computing an *approximate* solution is as hard as computing the exact solution (and hence cannot be done efficiently unless $\mathbf{P} = \mathbf{NP}$.) We show the equivalence of these two views in Section 1.3.

In Section 1.4 we demonstrate the usefulness of the **PCP** Theorem by using it to derive a very strong hardness of approximation result for the INDSET and MIN-VERTEX-COVER problems.

Although only one result is known as *the* **PCP** Theorem (Theorem 1.5 below) several related "**PCP** theorems" have been discovered, differing in various setting of parameters. In this chapter we prove such a theorem (Theorem 1.19 in Section 1.5) giving a weaker —but still useful— result than the full-fledged **PCP** Theorem. Another motivation for showing Theorem 1.19 is that it will play a part in the proof of the **PCP** Theorem, which appears in full in Chapter 22.

The various **PCP** theorems have revolutionized our understanding of the approximability of **NP**-hard problems. Chapter 22 will surveys several of these theorems.

11.1 Motivation: approximate solutions to NP-hard optimization problems

As mentioned in Chapter 2, one of the main motivations for the theory of **NP**-completeness was to understand the computational complexity of computing optimum solutions to combinatorial problems such as TSP or INDSET. Since $\mathbf{P} \neq \mathbf{NP}$ implies that thousands of **NP**-hard optimization problems do not have efficient algorithms, attention then focused on whether or not they have efficient *approximation algorithms*. In many practical settings, obtaining an approximate solution to a problem may be almost as good as solving it exactly, and could be a lot easier. Researchers are therefore interested in finding the best possible approximation algorithms for **NP**-hard optimization problems. For instance, it would be good to understand whether or not we could approximate interesting **NP**-problema within an arbitrary precision: if we could, then $\mathbf{P} \neq \mathbf{NP}$ would not be such a a big deal in practice. Many researchers suspected that there are inherent limits to approximation, and proving such limits was the main motivation behind the discovery of the **PCP** theorem.

In this section we illustrate the notion of approximation algorithms with an example. Let MAX-3SAT be the problem of finding, given a 3CNF Boolean formula φ as input, an assignment that maximizes the number of satisfied clauses. This problem is of course **NP**-hard, because the corresponding decision problem, 3SAT, is **NP**-complete. We define an approximation algorithm for MAX-3SAT in the following way.

Definition 11.1 *(Approximation of* MAX-3SAT*)*
For every 3CNF formula φ, the *value* of φ, denoted by $\mathsf{val}(\varphi)$, is the maximum fraction of clauses that can be satisfied by any assignment to φ's variables. In particular, φ is satisfiable iff $\mathsf{val}(\varphi) = 1$.
For every $\rho \leq 1$, an algorithm A is a *ρ-approximation* algorithm for MAX-3SAT if for every 3CNF formula φ with m clauses, $A(\varphi)$ outputs an assignment satisfying at least $\rho \cdot \mathsf{val}(\varphi)m$ of φ's clauses.

Now we give two simple examples of approximation algorithms; see the Chapter notes for references to more nontrivial algorithms.

Example 11.2 *(1/2-approximation for* MAX-3SAT*)*
We describe a polynomial-time algorithm that computes a 1/2-approximation for MAX-3SAT. The algorithm assigns values to the variables one by one in a greedy fashion, whereby the ith variable is assigned the value that results in satisfying at least 1/2 the clauses in which it appears. Any clause that gets satisfied is removed and not considered in assigning values to the remaining variables. Clearly, the final assignment will satisfy at least 1/2 of all clauses, which is certainly at least half of the maximum that the optimum assignment could satisfy.
Using semidefinite programming one can also design a polynomial-time $(7/8-\epsilon)$-approximation algorithm for every $\epsilon > 0$ (see chapter notes). Obtaining such a ratio is trivial if we restrict ourselves to $3CNF$ formulae with three distinct variables in each clause. Then a random assignment has probability 7/8 to satisfy it. Linearity of expectations (Claim A.3) implies that a random assignment is expected to satisfy a 7/8 fraction of the clauses. This observation can be turned into a simple probabilistic or even deterministic 7/8-approximation algorithm (see Exercise 1.3).

For a few problems, one can even design $(1-\epsilon)$-approximation algorithms for *every* $\epsilon > 0$. Exercise 1.12 asks you to show this for the **NP**-complete *knapsack* problem.

Example 11.3 *(1/2-approximation for* MIN-VERTEX-COVER*)*
The decision problem VERTEX-COVER was introduced in Example 2.15 in Chapter 2. The optimization version is MIN-VERTEX-COVER, in which we are given a graph and wish to determine the size of the minimum vertex cover (which, recall, is a set of vertices such that every graph edge contains one these vertices). For $\rho \leq 1$, a ρ-approximation algorithm for MIN-VERTEX-COVER is an algorithm that on input a graph G outputs a vertex cover whose size is at most $1/\rho$ times the size of the minimum vertex cover.[1] We now show a 1/2-approximation algorithm for MIN-VERTEX-COVER:
Start with $S \leftarrow \emptyset$. Pick any edge in the graph e_1, and add both its endpoints to S. Delete these two vertices from the graph as well as all edges adjacent to them. Iterate this process, picking edges $e_2, e_3, \ldots$ and adding their endpoints to S until you arrive at the empty graph.
Clearly, the set S at the end is such that every graph edge has an endpoint in S. Thus S is a vertex cover. Furthermore, the sequence of edges $e_1, e_2, \ldots$ used to build up S are pairwise disjoint; in other words, they form a *matching*. The cardinality of S is twice the number of edges in this matching. Furthermore, the minimum vertex cover must include at least one endpoint of each matching edge. Thus the cardinality of S is at most twice the cardinality of the minimum vertex cover.

11.2 Two views of the PCP Theorem.

The **PCP** Theorem can be viewed in two alternative ways, and internalizing both these ways is crucial to understanding both the theorem and its proof. One view of this theorem is that it talks about new, extremely robust proof systems. The other is that it talks about approximating combinatorial optimization problems.

11.2.1 PCP Theorem and locally testable proofs

The first view of the **PCP** Theorem (and the reason for its name) is as providing a new kind of proof systems. Suppose someone wants to convince you that a Boolean formula is satisfiable. He could present the usual certificate, namely, a satisfying assignment, which you could then check by substituting back into the formula. However, doing this requires reading the entire certificate. The **PCP** Theorem shows an interesting alternative: this person can easily rewrite his certificate so you can verify it by probabilistically selecting a constant number of locations—as low as 3 bits— to examine in it. Furthermore, this probabilistic verification has the following properties: **(1)** A correct certificate will never fail to convince you (that is, no choice of your random coins will make you reject it) and **(2)** If the formula is unsatisfiable, then you are guaranteed to reject *every claimed certificate* with high probability.

Of course, since Boolean satisfiability is **NP**-complete, every other **NP** language can be deterministically and efficiently reduced to it. Thus the **PCP** Theorem applies to every **NP** language. We mention one counterintuitive consequence. Let $\mathcal{A}$ be any one of the usual

[1] Many texts call such an algorithm a $1/\rho$-*approximation algorithm* instead.

axiomatic systems of mathematics for which proofs can be verified by a deterministic TM in time that is polynomial in the length of the proof. Recall the following language is in **NP**:

$$L = \{\langle \varphi, 1^n \rangle : \varphi \text{ has a proof in } \mathcal{A} \text{ of length } \leq n\}.$$

The **PCP** Theorem asserts that L has probabilistically checkable certificates. Such certificate can be viewed as an alternative notion of "proof" for mathematical statements that is just as valid as the usual notion. However, unlike standard mathematical proofs, where every line of the proof has to be checked to verify its validity, this new notion guarantees that proofs are probabilistically checkable by examining only a constant number of bits in them.[2]

We now make a more formal definition. Recall that a language L is in **NP** if there is a poly-time Turing machine V ("verifier") that, given input x, checks certificates (or membership proofs) to the effect that $x \in L$ (see Definition 2.1). In other words,

$$\begin{aligned} x \in L &\Rightarrow \exists \pi \text{ s.t. } V^\pi(x) = 1 \\ x \notin L &\Rightarrow \forall \pi \quad V^\pi(x) = 0, \end{aligned}$$

where V^π denotes "a verifier with access to certificate π".

The class **PCP** is a generalization of this notion, with the following changes. First, the verifier is probabilistic. Second, the verifier has *random access* to the proof string Π. This means that each bit of the proof string can be independently *queried* by the verifier via a special *address tape*: if the verifier desires say the ith bit in the proof string, it writes i on the address tape and then receives the bit $\pi[i]$.[3] The definition of **PCP** treats *queries* to the proof as a precious resource, to be used sparingly. Note also that since the address size is *logarithmic* in the proof size, this model in principle allows a polynomial-time verifier to check membership proofs of exponential size.

Verifiers can be *adaptive* or *nonadaptive*. A nonadaptive verifier selects its queries based only on its input and random tape. In other words, no query depends upon the responses to any of the prior queries. By contrast, an adaptive verifier can rely upon bits it has already queried in π to select its next queries. We restrict verifiers to be *nonadaptive*, since most **PCP** Theorems can be proved using nonadaptive verifiers, but Exercise 1.2 explores the power of adaptive queries.

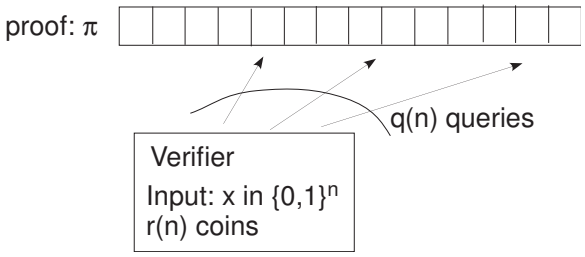

Figure 11.1 A **PCP** verifier for a language L gets an input x and random access to a string π. If $x \in L$ then there exists a string π that makes the verifier accepts, while if $x \notin L$ then the verifier rejects *every* proof π with probability at least $1/2$.

[2] One newspaper article about the discovery of the **PCP** Theorem carried the headline "New shortcut found for long math proofs!"

[3] For a precise formalization, see Exercise 1.9 discussing RAM Turing machines or Section 5.5 discussing oracle Turing machines.

Definition 11.4 *(**PCP** verifier)*
Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-**PCP** *verifier* if there's a polynomial-time probabilistic algorithm V satisfying:

Efficiency: On input a string $x \in \{0,1\}^n$ and given random access to a string $\pi \in \{0,1\}^*$ (which we call the *proof*), V uses at most $r(n)$ random coins and makes at most $q(n)$ non-adaptive queries to locations of π (see Figure 1.1). Then it outputs "1"(for "accept") or "0" (for "reject"). We let $V^\pi(x)$ denote the random variable representing V's output on input x and with random access to π.

Completeness: If $x \in L$ then there exists a proof $\pi \in \{0,1\}^*$ such that $\Pr[V^\pi(x) = 1] = 1$. We call this string π the *correct proof* for x.

Soundness: If $x \notin L$ then for every proof $\pi \in \{0,1\}^*$, $\Pr[V^\pi(x) = 1] \leq 1/2$.

We say that a language L is in $\mathbf{PCP}(r(n), q(n))$ if there are some constants $c, d > 0$ such that L has a $(c \cdot r(n), d \cdot q(n))$-**PCP** verifier.

The **PCP** Theorem says that every **NP** language has a highly efficient **PCP** verifier:

Theorem 11.5 *(The **PCP** Theorem* [AS92, ALM+92]*)*
$\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.

Remark 11.6
Some notes are in order:

1. The soundness condition stipulates that if $x \notin L$ then the verifier has to reject *every* proof with probability at least 1/2. Establishing this is the most difficult part of the proof.

2. We may assume without loss of generality that proofs checkable by an (r, q)-verifier are of length at most $q2^r$, since such a verifier can look on at most this number of locations with nonzero probability.

3. The previous remark implies that $\mathbf{PCP}(r(n), q(n)) \subseteq \mathbf{NTIME}(2^{O(r(n))}q(n))$ since a nondeterministic machine could guess the proof in $2^{O(r(n))}q(n)$ time, and verify it deterministically by running the verifier for all $2^{O(r(n))}$ possible choices of its random coin tosses. If the verifier accepts for all these possible coin tosses then the nondeterministic machine accepts.

   As a special case, $\mathbf{PCP}(\log n, 1) \subseteq \mathbf{NTIME}(2^{O(\log n)}) = \mathbf{NP}$: this is the trivial direction of the **PCP** Theorem.

4. The statement of the **PCP** Theorem allows verifiers for different **NP** languages to use a different number of query bits (so long as this number is constant). However, since every **NP** language is polynomial-time reducible to SAT, all these numbers can be upper bounded by a universal constant, namely, the number of query bits required by a verifier for SAT.

5. The constant $1/2$ in the soundness requirement of Definition 1.4 is arbitrary, in the sense that changing it to any other positive constant smaller than 1 will not change the class of languages defined. Indeed, a **PCP** verifier with soundness $1/2$ that uses r coins and makes q queries can be converted into a **PCP** verifier using cr coins and cq queries with soundness 2^{-c} by just repeating its execution c times (see Exercise 1.1).

Example 11.7
To get a better sense for what a **PCP** proof system looks like, we sketch two nontrivial **PCP** systems:

1. The language GNI of pairs of non-isomorphic graphs is in $\mathbf{PCP}(poly(n), 1)$. Say the input for GNI is $\langle G_0, G_1 \rangle$, where G_0, G_1 have both n nodes. The verifier expects π to contain, for each labeled graph H with n nodes, a bit $\pi[H] \in \{0,1\}$ corresponding to whether $H \equiv G_0$ or $H \equiv G_1$ ($\pi[H]$ can be arbitrary if neither case holds). In other words, π is an (exponentially long) array of bits indexed by the (adjacency matrix representations of) all possible n-vertex graphs.

   The verifier picks $b \in \{0,1\}$ at random and a random permutation. She applies the permutation to the vertices of G_b to obtain an isomorphic graph, H. She queries the corresponding bit of π and accepts iff the bit is b.

   If $G_0 \not\equiv G_1$, then clearly a proof π can be constructed which makes the verifier accept with probability 1. If $G_1 \equiv G_2$, then the probability that any π makes the verifier accept is at most $1/2$.
2. The protocols in Chapter 8 can be used (see Exercise 1.7) to show that the *permanent* has **PCP** proof system with polynomial randomness and queries. Once again, the length of the proof will be exponential.

In fact, both of these results are a special case of the following theorem:

Theorem 11.8 *(Scaled-up* **PCP***,* [BFLS91, ALM+92, AS92]*)* $\mathbf{PCP}(\text{poly}(n), 1) = \mathbf{NEXP}$

Above we use $\mathbf{PCP}(\text{poly}(n), 1)$ to denote the class $\cup_{c \geq 1} \mathbf{PCP}(n^c, 1)$. Theorem 1.8 can be thought of us a "scaled-up" version of the **PCP** Theorem. We omit the proof which uses similar techniques to the original proof of the **PCP** Theorem and Theorem 8.19 ($\mathbf{IP} = \mathbf{PSPACE}$). ♢

11.2.2 PCP and Hardness of Approximation

Another view of the **PCP** Theorem is that it shows that for many **NP** optimization problems, computing *approximate* solutions is no easier than computing exact solutions.

For concreteness, we focus for now on MAX-3SAT. Until 1992, we did not know whether or not MAX-3SAT has a polynomial-time ρ-approximation algorithm for *every* $\rho < 1$. It turns out that the **PCP** Theorem means that the answer is NO (unless $\mathbf{P} = \mathbf{NP}$). The reason is that it can be equivalently stated as follows:

Theorem 11.9 *(***PCP** *Theorem: Hardness of Approximation view)*
There exists $\rho < 1$ such that for every $L \in \mathbf{NP}$ there is a polynomial-time function f mapping strings to (representations of) 3CNF formulae such that:

$$x \in L \Rightarrow \mathsf{val}(f(x)) = 1 \quad (1)$$

$$x \notin L \Rightarrow \mathsf{val}(f(x)) < \rho\,. \quad (2)$$

This immediately implies the following corollary:

Corollary 11.10 *There exists some constant $\rho < 1$ such that if there is a polynomial-time ρ-approximation algorithm for* MAX-3SAT *then* $\mathbf{P} = \mathbf{NP}$. ♢

Indeed, Theorem 1.9 shows for every $L \in \mathbf{NP}$, a way to convert a ρ-approximation algorithm A for MAX-3SAT into an algorithm deciding L, since (1) and (2) together imply that $x \in L$ iff $A(f(x))$ yields an assignment satisfying at least a ρ fraction of $f(x)$'s clauses.

Later, in Chapter 22, we show a stronger **PCP** Theorem by Håstad which implies that for every $\epsilon > 0$, if there is a polynomial-time $(7/8+\epsilon)$-approximation algorithm for MAX-3SAT then $\mathbf{P} = \mathbf{NP}$. Hence the approximation algorithm for this problem mentioned in Example 1.2 is very likely *optimal*. The **PCP** Theorem (and the other **PCP** theorems that followed it) imply a host of such *hardness of approximation* results for many important problems, often showing that known approximation algorithms are optimal unless $\mathbf{P} = \mathbf{NP}$.

Why doesn't the Cook-Levin reduction suffice to prove Theorem 1.9? The first thing one would try to prove Theorem 1.9 is the reduction from any **NP** language to 3SAT in the Cook-Levin Theorem (Theorem 2.10). Unfortunately, it doesn't give such an f because it does not satisfy property (2): Exercise 1.11 asks you to show that we can satisfy almost all of the clauses in the formulae produced by the reduction. (This is what Papadimitriou and Yannakakis referred to in their quote.) Hence $\mathsf{val}(\cdot)$ is almost 1 for these formulae whereas Theorem 1.9 requires that $\mathsf{val}(\cdot) < \rho$ in one case.

11.3 Equivalence of the two views

We now show the equivalence of the "proof view" and the "hardness of approximation view" of the **PCP** Theorem. That is, we show that Theorem 1.5 is equivalent to Theorem 1.9. To do so we introduce the notion of *constraint satisfaction problems* (CSP). This is a generalization of 3SAT that turns up in many applications and also plays an important role in the proof of the **PCP** Theorem. A CSP problem generalizes 3SAT by allowing clauses of arbitrary form (instead of just OR of literals), including those depending upon more than 3 variables.

Definition 11.11 *(Constraint satisfaction problems (CSP))*
If q is a natural number then a qCSP instance φ is a collection of functions $\varphi_1, \ldots, \varphi_m$ (called *constraints*) from $\{0,1\}^n$ to $\{0,1\}$ such that each function φ_i depends on at most q of its input locations. That is, for every $i \in [m]$ there exist $j_1, \ldots, j_q \in [n]$ and $f : \{0,1\}^q \to \{0,1\}$ such that $\varphi_i(\mathbf{u}) = f(u_{j_1}, \ldots, u_{j_q})$ for every $\mathbf{u} \in \{0,1\}^n$.
We say that an *assignment* $\mathbf{u} \in \{0,1\}^n$ *satisfies* constraint φ_i if $\varphi_i(\mathbf{u}) = 1$. The fraction of constraints satisfied by $\mathbf{u}$ is $\frac{\sum_{i=1}^m \varphi_i(\mathbf{u})}{m}$, and we let $\mathsf{val}(\varphi)$ denote the maximum of this value over all $\mathbf{u} \in \{0,1\}^n$. We say that φ is *satisfiable* if $\mathsf{val}(\varphi) = 1$. We call q the *arity* of φ.

Example 11.12
3SAT is the subcase of qCSP where $q = 3$, and the constraints are OR's of the involved literals.

Notes:

1. We define the *size* of a qCSP-instance φ to be m, the number of constraints it has. Because variables not used by any constraints are redundant, we always assume $n \leq qm$. Note that a qCSP instance over n variables with m constraints can be described using $O(mn^q 2^q)$ bits. (In all cases we are interested in, q will be a constant independent of n, m.)

2. The simple greedy approximation algorithm for 3SAT can be generalized for the MAX qCSP problem of maximizing the number of satisfied constraints in a given qCSP instance. For any qCSP instance φ with m constraints, this algorithm will output an assignment satisfying $\frac{\mathsf{val}(\varphi)}{2^q} m$ constraints.

11.3.1 Equivalence of theorems 1.5 and 1.9.

We now show the equivalence of the two formulations of the **PCP** Theorem (theorems 1.5 and 1.9) by showing that they are both equivalent to the **NP**-hardness of a certain gap version of qCSP.

Definition 11.13 *(Gap* CSP*)* For every $q \in \mathbb{N}, \rho \leq 1$, define ρ-GAPqCSP to be the problem of determining for a given qCSP-instance φ whether **(1)** $\mathsf{val}(\varphi) = 1$ (in which case we say φ is a YES instance of ρ-GAPqCSP) or **(2)** $\mathsf{val}(\varphi) < \rho$ (in which case we say φ is a NO instance of ρ-GAPqCSP) .

We say that ρ-GAPqCSP is **NP**-hard if there is a polynomial-time function f mapping strings to (representations of) qCSP instances satisfying:

Completeness $x \in L \Rightarrow \mathsf{val}(f(x)) = 1$

Soundness $x \notin L \Rightarrow \mathsf{val}(f(x)) < \rho$

Theorem 11.14 *There exist constants $q \in \mathbb{N}$, $\rho \in (0,1)$ such that ρ-GAPqCSP is **NP**-hard.*$\diamondsuit$

We now show that theorems 1.5, 1.9 and 1.14 are all equivalent to one another.

Theorem 1.5 implies Theorem 1.14. Assume that $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$. We will show that $1/2$-GAPqCSP is **NP**-hard for some constant q. It is enough to reduce a single **NP**-complete language such as 3SAT to $1/2$-GAPqCSP for some constant q. Under our assumption, 3SAT has a **PCP** system in which the verifier V makes a constant number of queries, which we denote by q, and uses $c \log n$ random coins for some constant c. Given every input x and $r \in \{0,1\}^{c \log n}$, define $V_{x,r}$ to be the function that on input a proof π outputs 1 if the verifier will accept the proof π on input x and coins r. Note that $V_{x,r}$ depends on at most q locations. Thus for every $x \in \{0,1\}^n$, the collection $\varphi = \{V_{x,r}\}_{r \in \{0,1\}^{c \log n}}$ is a polynomial-sized qCSP instance. Furthermore, since V runs in polynomial-time, the transformation of x to φ can also be carried out in polynomial-time. By the completeness and soundness of the **PCP** system, if $x \in$ 3SAT then φ will satisfy $\mathsf{val}(\varphi) = 1$, while if $x \notin$ 3SAT then φ will satisfy $\mathsf{val}(\varphi) \leq 1/2$. ■

Theorem 1.14 implies Theorem 1.5. Suppose that ρ-GAPqCSP is **NP**-hard for some constants q,$\rho < 1$. Then this easily translates into a **PCP** system with q queries, ρ soundness and logarithmic randomness for any language L: given an input x, the verifier will run the reduction $f(x)$ to obtain a qCSP instance $\varphi = \{\varphi_i\}_{i=1}^m$. It will expect the proof π to be an assignment to the variables of φ, which it will verify by choosing a random $i \in [m]$ and checking that φ_i is satisfied (by making q queries). Clearly, if $x \in L$ then the verifier will accept with probability 1, while if $x \notin L$ it will accept with probability at most ρ. The soundness can be boosted to $1/2$ at the expense of a constant factor in the randomness and number of queries (see Exercise 1.1). ■

Theorem 1.9 is equivalent to Theorem 1.14. Since $3CNF$ formulas are a special case of 3CSP instances, Theorem 1.9 implies Theorem 1.14. We now show the other direction.

Let $\epsilon > 0$ and $q \in \mathbb{N}$ be such that by Theorem 1.14, $(1-\epsilon)$-GAPqCSP is **NP**-hard. Let φ be a qCSP instance over n variables with m constraints. Each constraint φ_i of φ can be expressed as an AND of at most 2^q clauses, where each clause is the OR of at most q variables or their negations. Let φ' denote the collection of at most $m2^q$ clauses corresponding to all the constraints of φ. If φ is a YES instance of $(1-\epsilon)$-GAPqCSP (i.e., it is satisfiable) then there exists an assignment satisfying all the clauses of φ'. If φ is a NO instance of $(1-\epsilon)$-GAPqCSP then every assignment violates at least an ϵ fraction of the

constraints of φ and hence violates at least an $\frac{\epsilon}{2^q}$ fraction of the constraints of φ. We can use the Cook-Levin technique of Chapter 2 (Theorem 2.10), to transform any clause C on q variables $u_1, \ldots, u_q$ to a set $C_1, \ldots, C_q$ of clauses over the variables $u_1, \ldots, u_q$ and additional auxiliary variables $y_1, \ldots, y_q$ such that **(1)** each clause C_i is the OR of at most three variables or their negations, **(2)** if $u_1, \ldots, u_q$ satisfy C then there is an assignment to $y_1, \ldots, y_q$ such that $u_1, \ldots, u_q, y_1, \ldots, y_q$ simultaneously satisfy $C_1, \ldots, C_q$ and **(3)** if $u_1, \ldots, u_q$ does not satisfy C then for every assignment to $y_1, \ldots, y_q$, there is some clause C_i that is not satisfied by $u_1, \ldots, u_q, y_1, \ldots, y_q$.

Let φ'' denote the collection of at most $qm2^q$ clauses over the $n + qm$ variables obtained in this way from φ'. Note that φ'' is a 3SAT formula. Our reduction will map φ to φ''. Completeness holds since if φ were satisfiable then so would be φ' and hence also φ''. Soundness holds since if every assignment violates at least an ϵ fraction of the constraints of φ, then every assignment violates at least an $\frac{\epsilon}{2^q}$ fraction of the constraints of φ', and so every assignment violates at least an $\frac{\epsilon}{q2^q}$ fraction of the constraints of φ'' ■

11.3.2 Review of the two views of the PCP Theorem

It is worthwhile to review this very useful equivalence between the "proof view" and the "hardness of approximation view" of the **PCP** Theorem:

Proof view		Hardness of approximation view
PCP verifier (V)	$\longleftrightarrow$	CSP instance (φ)
PCP proof (π)	$\longleftrightarrow$	Assignment to variables ($\mathbf{u}$)
Length of proof	$\longleftrightarrow$	Number of variables (n)
Number of queries (q)	$\longleftrightarrow$	Arity of constraints (q)
Number of random bits (r)	$\longleftrightarrow$	Logarithm of number of constraints ($\log m$)
Soundness parameter (typically 1/2)	$\longleftrightarrow$	Maximum of $\mathsf{val}(\varphi)$ for a NO instance
Theorem 1.5 ($\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$)	$\longleftrightarrow$	Theorem 1.14 (ρ-GAPqCSP is **NP**-hard) , Theorem 1.9 (MAX-3SAT is **NP**-hard to ρ-approximate)

Table 11.1 Two views of the **PCP** Theorem.

11.4 Hardness of approximation for vertex cover and independent set

The **PCP** Theorem implies hardness of approximation results for many more problems than just 3SAT and CSP. As an example we show a hardness of approximation result for the maximum independent set (MAX-INDSET) problem we encountered in Chapter 2 (Example 2.2) and for the MIN-VERTEX-COVER problem encountered in Example 1.3. Note that the inapproximability result for MAX-INDSET is stronger than the result for MIN-VERTEX-COVER.

Theorem 11.15 *There is some $\gamma < 1$ such that computing a γ-approximation to* MIN-VERTEX-COVER *is* **NP***-hard. For every $\rho < 1$, computing a ρ-approximation to* INDSET *is* **NP***-hard.* ♢

Since a vertex cover is a set of vertices touching all edges of the graph, its complement is an an independent set. Thus the two problems are trivially equivalent with respect to exact solution: the largest independent set is simply the complement of the smallest vertex cover. However, this does not imply that they are equivalent with respect to approximation. Denoting the size of the minimum vertex cover by VC and the size of the largest independent set by IS we see that VC $= n -$ IS. Thus a ρ-approximation for INDSET would produce an independent set of size $\rho \cdot$ IS, and if we wish to use this to compute an approximation to MIN-VERTEX-COVER then we obtain a vertex cover of size $n - \rho \cdot$ IS. This yields an

approximation ratio of $\frac{n-\rho \mathrm{IS}}{n-\mathrm{IS}}$ for MIN-VERTEX-COVER, which could be arbitrarily small if IS is close to n. In fact, Theorem 1.15 shows that if $\mathbf{P} \neq \mathbf{NP}$, then the approximability of the two problems is inherently different: we already saw that MIN-VERTEX-COVER has a polynomial-time 1/2-approximation algorithm (Example 2.2) whereas if $\mathbf{P} \neq \mathbf{NP}$ then INDSET does not have a ρ-approximation algorithm for *every* $\rho < 1$.

We first show using the **PCP** Theorem that there is some constant $\rho < 1$ such that both problems cannot be ρ-approximated in polynomial-time (unless $\mathbf{P} = \mathbf{NP}$). We then show how to "amplify" the approximation gap and make ρ as small as desired in case of INDSET.

Lemma 11.16 *There exist a polynomial-time computable transformation f from 3CNF formulae to graphs such that for every 3CNF formula φ, $f(\varphi)$ is an n-vertex graph whose largest independent set has size $\mathsf{val}(\varphi)\frac{n}{7}$.* ♢

PROOF SKETCH: We apply the "normal" **NP**-completeness reduction for INDSET (see proof of Theorem 2.15) on this 3CNF formula, and observe that it satisfies the desired property. We leave verifying the details as Exercise 1.5. ■

The following Corollary is immediate.

Corollary 11.17 *If $\mathbf{P} \neq \mathbf{NP}$ then are some constants $\rho < 1, \rho' < 1$ such that the problem* INDSET *cannot be ρ-approximated in polynomial time and* MIN-VERTEX-COVER *cannot be ρ'-approximated.* ♢

PROOF: Let L be any **NP** language. Theorem 1.9 implies that the decision problem for L can be reduced to approximating MAX-3SAT. Specifically, the reduction produces a 3CNF formula φ that is either satisfiable or satisfies $\mathsf{val}(\varphi) < \rho$, where $\rho < 1$ is some constant. Then we can apply the reduction of Lemma 1.16 on this 3CNF formula and conclude that a ρ-approximation to INDSET would allow us to do a ρ-approximation to MAX-3SAT on φ. Thus it follows that ρ-approximation to INDSET is **NP**-hard.

The result for MIN-VERTEX-COVER follows from the observation that the minimum vertex cover in the graph resulting from the reduction of the previous paragraph has size $n - \mathsf{val}(\varphi)\frac{n}{7}$. It follows that if MIN-VERTEX-COVER had a ρ'-approximation for $\rho' = 6/(7-\rho)$ then it would allow us to find a vertex cover of size $\frac{1}{\rho'}(n - \frac{n}{7})$ in the case $\mathsf{val}(\varphi) = 1$, and this size is at most $n - \rho n/7$. Thus we conclude that such an approximation would allow us to distinguish the cases $\mathsf{val}(\varphi) = 1$ and $\mathsf{val}(\varphi) < \rho$, which by Theorem 1.9 is **NP**-hard. ■

To complete the proof of Theorem 1.15 we need to amplify this approximation gap for INDSET. Such amplification is possible for many combinatorial problems thanks to a certain "self-improvement" property. In case of INDSET, a simple self-improvement is possible using a *graph product*.

For any n-vertex graph G, define G^k to be a graph on $\binom{n}{k}$ vertices whose vertices correspond to all subsets of vertices of G of size k Two subsets S_1, S_2 are adjacent if $S_1 \cup S_2$ is an independent set in G. It is easily checked that the largest independent of G^k corresponds to all k-size subsets of the a largest independent set in G, and therefore has size $\binom{\mathrm{IS}}{k}$, where IS is the size of the largest independent set in G. Thus if we take the graph produced by the reduction of Corollary 1.17 and take its k-wise product, then the size of the largest independent set differs in the two cases by a factor ρ^k. Choosing k large enough, ρ^k can be made smaller than any desired constant. The running time of the reduction becomes n^k, which is polynomial for every fixed k. ■

Remark 11.18 *(Levin reductions)*
In Chapter 2, we defined L' to be **NP**-hard if every $L \in \mathbf{NP}$ reduces to L'. The reduction was a polynomial-time function f such that $x \in L \Leftrightarrow f(x) \in L'$. In all cases, we proved that $x \in L \Rightarrow f(x) \in L'$ by showing a way to map a *certificate* for the fact that $x \in L$ to a certificate for the fact that $x' \in L'$. Although the definition of a Karp reduction does not require that this mapping is efficient, this was often the case. Similarly we proved that $f(x) \in L' \Rightarrow x \in L$ by showing a way to map a certificate for the fact that $x' \in L'$ to a certificate for the fact that $x \in L$. Again the proofs typically yield an efficient way to compute this mapping. We call reductions with these properties "Levin reductions" (see

the proof of Theorem 2.18). It is worthwhile to observe that the **PCP** reductions of this chapter also satisfy this property. For example, the proof of Theorem 1.16 actually yields a way not just to map, say, a CNF formula φ into a graph G such that φ is satisfiable iff G has a "large" independent set, but actually shows how to efficiently map a satisfying assignment for φ into a large independent set in G and a not-too-small independent set in G into a satisfying assignment for φ. This will become clear from our proof of the **PCP** Theorem in Chapter 22.

11.5 NP ⊆ PCP(poly(n), 1): PCP from the Walsh-Hadamard code

We now prove a weaker version of the **PCP** Theorem, showing that every **NP** statement has an exponentially-long proof that can be locally tested by only looking at a constant number of bits. In addition to giving a taste of how one proves **PCP** Theorems, techniques from this section will be used in the proof of the full-fledged **PCP** theorem in Chapter 22.

Theorem 11.19 *(Exponential-sized* **PCP** *system for* **NP** [ALM+92]*)*
NP ⊆ **PCP**(poly(n), 1)

We prove this theorem by designing an appropriate verifier for an **NP**-complete language. The verifier expects the proof to contain an encoded version of the usual certificate. The verifier checks such an encoded certificate by simple probabilistic tests.

11.5.1 Tool: Linearity Testing and the Walsh-Hadamard Code

We use the *Walsh-Hadamard code* (see also Section 19.2.2, though the treatment here is self-contained). It is a way to encode bit strings of length n by *linear functions* in n variables over GF(2). The encoding function $\mathsf{WH} : \{0,1\}^* \to \{0,1\}^*$ maps a string $\mathbf{u} \in \{0,1\}^n$ to the truth table of the function $\mathbf{x} \mapsto \mathbf{u} \odot \mathbf{x}$, where for $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ we define $\mathbf{x} \odot \mathbf{y} = \sum_{i=1}^{n} x_i y_i \pmod 2$. Note that this is a very inefficient encoding method: an n-bit string $\mathbf{u} \in \{0,1\}^n$ is encoded using $|\mathsf{WH}(\mathbf{u})| = 2^n$ bits. If $f \in \{0,1\}^{2^n}$ is equal to $\mathsf{WH}(\mathbf{u})$ for some $\mathbf{u}$ then we say that f is a *Walsh-Hadamard codeword*. Such a string $f \in \{0,1\}^{2^n}$ can also be viewed as a function from $\{0,1\}^n$ to $\{0,1\}$.

Below, we repeatedly use the following fact (see Claim A.31):

RANDOM SUBSUM PRINCIPLE: *If* $\mathbf{u} \neq \mathbf{v}$ *then for* $1/2$ *the choices of* $\mathbf{x}$, $\mathbf{u} \odot \mathbf{x} \neq \mathbf{v} \odot \mathbf{x}$.

The random subsum principle implies that the Walsh-Hadamard code is an error correcting code with minimum distance $1/2$, by which we mean that for every $\mathbf{u} \neq \mathbf{v} \in \{0,1\}^n$, the encodings $\mathsf{WH}(\mathbf{u})$ and $\mathsf{WH}(\mathbf{v})$ differ in at least half the bits. Now we talk about local tests for the Walsh-Hadamard code (i.e., tests making only $O(1)$ queries).

Local testing of Walsh-Hadamard code. Suppose we are given access to a function $f : \{0,1\}^n \to \{0,1\}$ and want to *test* whether or not f is actually a codeword of Walsh-Hadamard. Since the Walsh-Hadamard codewords are precisely the set of all *linear* functions from $\{0,1\}^n$ to $\{0,1\}$, we can test f by checking that

$$f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}) \tag{3}$$

for all the 2^{2n} pairs $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ (where "+" on the left side of (3) denotes vector addition over $\mathrm{GF}(2)^n$ and on the right side denotes addition over GF(2)). This test works by definition, but it involves reading all 2^n values of f.

Can we test f by reading only a *constant* number of its values? The natural test is to choose $\mathbf{x}, \mathbf{y}$ at random and verify (3). Clearly, even such a local test accepts a linear function with probability 1. However, now we can no longer guarantee that every function that is not linear is rejected with high probability! For example, if f is very close to being a linear function, meaning that f is obtained by modifying a linear function on a very small fraction of its inputs, then such a *local* test will encounter the nonlinear part with very low probability, and thus not be able to distinguish f from a linear function. So we set our goal less ambitiously: a test that on one hand accepts every linear function, and on the other hand rejects with high probability every function that is *far from linear*. The natural test above suffices for this job.

Definition 11.20 Let $\rho \in [0,1]$. We say that $f, g : \{0,1\}^n \to \{0,1\}$ are *ρ-close* if $\Pr_{\mathbf{x} \in_R \{0,1\}^n}[f(\mathbf{x}) = g(\mathbf{x})] \geq \rho$. We say that f is *ρ-close to a linear function* if there exists a linear function g such that f and g are ρ-close. ♢

Theorem 11.21 *(Linearity Testing* [BLR90]*) Let $f : \{0,1\}^n \to \{0,1\}$ be such that*

$$\Pr_{\mathbf{x},\mathbf{y} \in_R \{0,1\}^n}[f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})] \geq \rho$$

for some $\rho > 1/2$. Then f is ρ-close to a linear function. ♢

We defer the proof of Theorem 1.21 to Section 22.5 of Chapter 22. For every $\delta \in (0, 1/2)$, we can obtain a linearity test that rejects with probability at least $1/2$ every function that is not $(1–\delta)$-close to a linear function, by testing Condition (3) repeatedly $O(1/\delta)$ times with independent randomness. We call such a test a *$(1–\delta)$-linearity test*.

Local decoding of Walsh-Hadamard code. Suppose that for $\delta < \frac{1}{4}$ the function $f : \{0,1\}^n \to \{0,1\}$ is $(1–\delta)$-close to some linear function $\tilde{f}$. Because every two linear functions differ on half of their inputs, the function $\tilde{f}$ is uniquely determined by f. Suppose we are given $\mathbf{x} \in \{0,1\}^n$ and random access to f. Can we obtain the value $\tilde{f}(\mathbf{x})$ using only a constant number of queries? The naive answer is that since most $\mathbf{x}$'s satisfy $f(\mathbf{x}) = \tilde{f}(\mathbf{x})$, we should be able to learn $\tilde{f}(\mathbf{x})$ with good probability by making only the single query $\mathbf{x}$ to f. The problem is that $\mathbf{x}$ could very well be one of the places where f and $\tilde{f}$ differ. Fortunately, there is still a simple way to learn $\tilde{f}(\mathbf{x})$ while making only two queries to f:

1. Choose $\mathbf{x}' \in_R \{0,1\}^n$.
2. Set $\mathbf{x}'' = \mathbf{x} + \mathbf{x}'$.
3. Let $\mathbf{y}' = f(\mathbf{x}')$ and $\mathbf{y}'' = f(\mathbf{x}'')$.
4. Output $\mathbf{y}' + \mathbf{y}''$.

Since both $\mathbf{x}'$ and $\mathbf{x}''$ are individually uniformly distributed (even though they are dependent), by the union bound with probability at least $1 - 2\delta$ we have $\mathbf{y}' = \tilde{f}(\mathbf{x}')$ and $\mathbf{y}'' = \tilde{f}(\mathbf{x}'')$. Yet by the linearity of $\tilde{f}$, $\tilde{f}(\mathbf{x}) = \tilde{f}(\mathbf{x}' + \mathbf{x}'') = \tilde{f}(\mathbf{x}') + \tilde{f}(\mathbf{x}'')$, and hence with at least $1 - 2\delta$ probability $\tilde{f}(\mathbf{x}) = \mathbf{y}' + \mathbf{y}''$. (We use here the fact that over GF(2), $a + b = a - b$.) This technique is called *local decoding* of the Walsh-Hadamard code since it allows to recover any bit of the correct codeword (the linear function $\tilde{f}$) from a corrupted version (the function f) while making only a constant number of queries. It is also known as *self correction* of the Walsh-Hadamard code.

11.5.2 Proof of Theorem 1.19

We will show a $(\text{poly}(n), 1)$-verifier proof system for a particular **NP**-complete language L. The result that $\mathbf{NP} \subseteq \mathbf{PCP}(\text{poly}(n), 1)$ follows since every **NP** language is reducible to L. The **NP**-complete language L we use is QUADEQ, the language of systems of quadratic equations over GF(2) $= \{0,1\}$ that are satisfiable.

Example 11.22

The following is an instance of QUADEQ over the variables $u_1, \ldots, u_5$:

$$\begin{aligned} u_1u_2 + u_3u_4 + u_1u_5 &= 1 \\ u_2u_3 + u_1u_4 &= 0 \\ u_1u_4 + u_3u_5 + u_3u_4 &= 1 \end{aligned}$$

This instance is satisfiable since the all-1 assignment satisfies all the equations.

QUADEQ is **NP**-complete, as can be checked by reducing from the **NP**-complete language CKT-SAT of satisfiable Boolean circuits (see Section 6.1.2). The idea is to have a variable represent the value of each wire in the circuit (including the input wires) and to express AND and OR using the equivalent quadratic polynomial: $x \vee y = 1$ iff $(1-x)(1-y) = 0$, and so on. Details are left as Exercise 1.15.

Since $u_i = (u_i)^2$ in GF(2), we can assume the equations do not contain terms of the form u_i (i.e., all terms are of degree exactly two). Hence m quadratic equations over the variables $u_1, \ldots, u_n$ can be described by an $m \times n^2$ matrix A and an m-dimensional vector $\mathbf{b}$ (both over GF(2)). Indeed, the problem QUADEQ can be phrased as the task, given such $A, \mathbf{b}$, of finding an n^2-dimensional vector U satisfying **(1)** $AU = \mathbf{b}$ and **(2)** U is the *tensor product* $\mathbf{u} \otimes \mathbf{u}$ of some n-dimensional vector $\mathbf{u}$.[4]

Figure 11.2 The **PCP** proof that a set of quadratic equations is satisfiable consists of WH(u) and WH($u \otimes u$) for some vector u. The verifier first checks that the proof is close to having this form, and then uses the local decoder of the Walsh-Hadamard code to ensure that u is a solution for the quadratic equation instance. In the figure above the dotted areas represent corrupted coordinates.

The PCP verifier. We now describe the **PCP** system for QUADEQ. Let $A, \mathbf{b}$ be an instance of QUADEQ and suppose that $A, \mathbf{b}$ is satisfiable by an assignment $\mathbf{u} \in \{0,1\}^n$. The verifier V gets access to a proof $\pi \in \{0,1\}^{2^n + 2^{n^2}}$, which we interpret as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$. In the correct **PCP** proof π for A, b, the function f will be the Walsh-Hadamard encoding for $\mathbf{u}$ and the function g will be the Walsh-Hadamard encoding for $\mathbf{u} \otimes \mathbf{u}$. That is, we will design the **PCP** verifier V in a way ensuring that it accepts proofs of this form with probability one, hence satisfying the completeness condition. The analysis repeatedly uses the random subsum principle.

Step 1: Check that f, g are linear functions. As already noted, this isn't something that the verifier can check per se using local tests. Instead, the verifier performs a 0.999-linearity test on both f, g, and rejects the proof at once if either test fails.

Thus, if either of f, g is not 0.999-close to a linear function, then V rejects with high probability. Therefore for the rest of the procedure we can assume that there exist two linear functions $\tilde{f} : \{0,1\}^n \to \{0,1\}$ and $\tilde{g} : \{0,1\}^{n^2} \to \{0,1\}$ such that $\tilde{f}$ is 0.999-close to f, and $\tilde{g}$ is 0.999-close to g. (Note: in a correct proof, the tests succeed with probability 1 and $\tilde{f} = f$ and $\tilde{g} = g$.)

In fact, we will assume that for Steps 2 and 3, the verifier can query $\tilde{f}, \tilde{g}$ at any desired point. The reason is that local decoding allows the verifier to recover any desired value of $\tilde{f}, \tilde{g}$ with good probability, and Steps 2 and 3 will only use a small (less than 20) number

[4] If $\mathbf{x}, \mathbf{y}$ are two n-dimensional vectors then their *tensor product*, denoted $\mathbf{x} \otimes \mathbf{y}$, is the n^2-dimensional vector (or $n \times n$ matrix) whose $\langle i, j \rangle^{th}$ entry is $x_i y_j$ (identifying $[n^2]$ with $[n] \times [n]$ in some canonical way). See also Section 21.3.3.

of queries to $\tilde{f}, \tilde{g}$. Thus with high probability (say > 0.9) local decoding will succeed on all these queries.

NOTATION: To simplify notation and in the rest of the procedure we use f, g for $\tilde{f}, \tilde{g}$ respectively. (This is OK since as argued above, V can query $\tilde{f}, \tilde{g}$ at will.) In particular we assume both f and g are linear, and thus they must encode some strings $\mathbf{u} \in \{0,1\}^n$ and $\mathbf{w} \in \{0,1\}^{n^2}$. In other words, f, g are the functions given by $f(\mathbf{r}) = \mathbf{u} \odot \mathbf{r}$ and $g(\mathbf{z}) = \mathbf{w} \odot \mathbf{z}$.

Step 2: Verify that g encodes $\mathbf{u} \otimes \mathbf{u}$, where $\mathbf{u} \in \{0,1\}^n$ is the string encoded by f. The verifier does the following test ten times using independent random bits: "Choose $\mathbf{r}, \mathbf{r}'$ independently at random from $\{0,1\}^n$, and if $f(\mathbf{r})f(\mathbf{r}') \neq g(\mathbf{r} \otimes \mathbf{r}')$ then halt and reject."

In a correct proof, $\mathbf{w} = \mathbf{u} \otimes \mathbf{u}$, so

$$f(\mathbf{r})f(\mathbf{r}') = \left(\sum_{i \in [n]} u_i r_i\right)\left(\sum_{j \in [n]} u_j r'_j\right) = \sum_{i,j \in [n]} u_i u_j r_i r'_j = (\mathbf{u} \otimes \mathbf{u}) \odot (\mathbf{r} \otimes \mathbf{r}'),$$

which in the correct proof is equal to $g(\mathbf{r} \otimes \mathbf{r}')$. Thus Step 2 never rejects a correct proof.

Suppose now that, unlike the case of the correct proof, $\mathbf{w} \neq \mathbf{u} \otimes \mathbf{u}$. We claim that in each of the ten trials V will halt and reject with probability at least $\frac{1}{4}$. (Thus the probability of rejecting in at least one trial is at least $1 - (3/4)^{10} > 0.9$.) Indeed, let W be an $n \times n$ matrix with the same entries as $\mathbf{w}$, let U be the $n \times n$ matrix such that $U_{i,j} = u_i u_j$ and think of $\mathbf{r}$ as a row vector and $\mathbf{r}'$ as a column vector. In this notation,

$$g(\mathbf{r} \otimes \mathbf{r}') = \mathbf{w} \odot (\mathbf{r} \otimes \mathbf{r}') = \sum_{i,j \in [n]} w_{i,j} r_i r'_j = \mathbf{r} W \mathbf{r}'$$

$$f(\mathbf{r})f(\mathbf{r}') = (\mathbf{u} \odot \mathbf{r})(\mathbf{u} \odot \mathbf{r}') = (\sum_{i=1}^{n} u_i r_i)(\sum_{j=1}^{n} u_j r'_j) = \sum_{i,j \in [n]} u_i u_j r_i r'_j = \mathbf{r} U \mathbf{r}'$$

And V rejects if $\mathbf{r}W\mathbf{r}' \neq \mathbf{r}U\mathbf{r}'$. The random subsum principle implies that if $W \neq U$ then at least $1/2$ of all $\mathbf{r}$ satisfy $\mathbf{r}W \neq \mathbf{r}U$. Applying the random subsum principle for each such $\mathbf{r}$, we conclude that at least $1/2$ the $\mathbf{r}'$ satisfy $\mathbf{r}W\mathbf{r}' \neq \mathbf{r}U\mathbf{r}'$. We conclude that the trial rejects for at least $1/4$ of all pairs $\mathbf{r}, \mathbf{r}'$.

Step 3: Verify that f encodes a satisfying assignment. Using all that has been verified about f, g in the previous two steps, it is easy to check that any particular equation, say the kth equation of the input, is satisfied by $\mathbf{u}$, namely,

$$\sum_{i,j} A_{k,(i,j)} u_i u_j = b_k. \tag{4}$$

Denoting by z the n^2 dimensional vector $(A_{k,(i,j)})$ (where i, j vary over $[1..n]$), we see that the left hand side is nothing but $g(\mathbf{z})$. Since the verifier knows $A_{k,(i,j)}$ and b_k, it simply queries g at $\mathbf{z}$ and checks that $g(\mathbf{z}) = b_k$.

The drawback of the above idea is that in order to check that $\mathbf{u}$ satisfies the entire system, the verifier needs to make a query to g for each $k = 1, 2, \ldots, m$, whereas the number of queries is required to be independent of m. Luckily, we can use the random subsum principle again! The verifier takes a random subset of the equations and computes their sum mod 2. (In other words, for $k = 1, 2, \ldots, m$ multiply the equation in (4) by a random bit and take the sum.) This sum is a new quadratic equation, and the random subsum principle implies that if $\mathbf{u}$ does not satisfy even one equation in the original system, then with probability at least $1/2$ it will not satisfy this new equation. The verifier checks that $\mathbf{u}$ satisfies this new equation.

Overall, we get a verifier V such that **(1)** if $A, \mathbf{b}$ is satisfiable then V accepts the correct proof with probability 1 and **(2)** if $A, \mathbf{b}$ is not satisfiable then V accepts every proof with probability at most 0.8. The probability of accepting a proof for a false statement can be reduced to $1/2$ by simple repetition, completing the proof of Theorem 1.19. ■

Chapter notes and history

The notion of approximation algorithms predates the discovery of **NP**-completeness; for instance Graham's 1966 paper [Gra66] already gives an approximation algorithm for a scheduling problem that was later proven **NP**-complete. Shortly after the discovery of **NP**-completeness, Johnson [Joh74] formalized the issue of computing approximate solutions, gave some easy approximation algorithms (such as the 1/2-approximation for MAX-SAT) for a variety of problems and posed the question of whether better algorithms exist. Over the next 20 years, although a few results were proven regarding the hardness of computing approximate solutions (such as for general TSP in Sahni and Gonzalez [SG76]) and a few approximation algorithms were designed, it became increasingly obvious that we were lacking serious techniques for proving the hardness of approximation. One of the problems seemed to be that there were no obvious interreducibilies among problems that preserved approximability. The paper by Papadimitriou and Yannakakis [PY88] showed such interreducibilities among a large set of problems they called MAX-SNP, and showed further that MAX-3SAT is *complete* for this class. This made MAX-3SAT an attractive problem to study both from point of view of algorithm design and for proving hardness results.

Soon after this work, seemingly unrelated developments occurred in study of interactive proofs, some of which we studied in Chapter 8. The most relevant for the topic of this chapter was the result of Babai, Fortnow and Lund [BFL90] that **MIP** = **NEXP**, which was soon made to apply to **NP** in the paper of Babai, Fortnow, Levin, and Szegedy [BFLS91]. After this there were a sequence of swift developments. In 1991 came a stunning result by Feige, Goldwasser, Lovasz, Safra and Szegedy [FGL$^+$91] which showed that if SAT does not have subexponential time algorithms then the INDSET problem cannot be approximated within a factor $2^{\log^{1-\epsilon} n}$ for any $\epsilon > 0$. This was the first paper to connect hardness of approximation with **PCP**-like theorems, though at the time many researchers felt (especially because the result did not prove **NP**-completeness per se) that this was the "wrong idea" and the result would ultimately be reproven with no mention of interactive proofs. (Intriguingly, Dinur's gap amplification lemma brings us closer to that dream.) However, a year later Arora and Safra [AS92] further refined the ideas of [BFL90] (together with the idea of verifier composition) to prove that approximating INDSET is actually **NP**-complete. They also proved a surprising new characterization of **NP** in terms of **PCP**, namely, **NP** = **PCP**$(\log n, \sqrt{\log n})$. At that point it became clear that if the query parameter could be sublogarithmic, it might well be made a constant! The subsequent paper of Arora, Lund, Motwani, Sudan, and Szegedy [ALM$^+$92] took this next step (in the process introducing the constant-bit verifier of Section 1.5, as well as other ideas) to prove **NP** = **PCP**$(\log n, 1)$, which they showed also implied the **NP**-hardness of approximating MAX-3SAT. Since then many other **PCP** theorems have been proven, as surveyed in Chapter 22. (Note that in this chapter we derived the hardness result for INDSET from the result for MAX-3SAT, even though historically the former happened first.)

The overall idea in the AS-ALMSS proof of the **PCP** Theorem (as indeed the one in the proof of **MIP** = **NEXP**) is similar to the proof of Theorem 1.19. In fact Theorem 1.19 is the only part of the original proof that still survives in our writeup; the rest of the proof in Chapter 22 is a more recent proof due to Dinur. However, in addition to using encodings based upon the Walsh-Hadamard code the AS-ALMSS proof also used encodings based upon low degree multivariate polynomials. These have associated procedures analogous to the linearity test and local decoding, though the proofs of correctness are a fair bit harder. The proof also drew intuition from the topic of self-testing and self-correcting programs [BLR90, RS92].

The PCP Theorem led to a flurry of results about hardness of approximation. See Trevisan [Tre05] for a recent survey and Arora-Lund [AL95] for an older one.

The PCP Theorem, as well as its cousin, **MIP** = **NEXP**, does not relativize [FRS88].

In this chapter we only talked about some very trivial approximation algorithms, which are unfortunately not very representative of the state of the art. See Hochbaum [Hoc97] and Vazirani [Vaz01] for a good survey of the many ingenious approximation algorithms that have been developed.

Exercises

11.1 Prove that for every two functions $r, q : \mathbb{N} \to \mathbb{N}$ and constant $s < 1$, changing the constant in the soundness condition in Definition 1.4 from $1/2$ to s will not change the class **PCP**(r, q).

11.2 Prove that any language L that has a **PCP**-verifier using r coins and q *adaptive* queries also has a

standard (i.e., non-adaptive) verifier using r coins and 2^q queries.

11.3 Give a probabilistic polynomial-time algorithm that given a 3CNF formula φ with exactly three distinct variables in each clause, outputs an assignment satisfying at least a 7/8 fraction of φ's clauses. H455

11.4 Give a *deterministic* polynomial-time algorithm with the same approximation guarantee as in Exercise 1.3. H455

11.5 Prove Lemma 1.16.

11.6 Prove that $\mathbf{PCP}(0, \log n) = \mathbf{P}$. Prove that $\mathbf{PCP}(0, \text{poly}(n)) = \mathbf{NP}$.

11.7 Let L be the language of pairs $\langle A, k\rangle$ such that A is a 0/1 matrix and $k \in \mathbb{Z}$ satisfying $\mathsf{perm}(A) = k$ (see Section 8.6.2). Prove that L is in $\mathbf{PCP}(\text{poly}(n), \text{poly}(n))$.

11.8 ([AS92]) Show that if $\mathsf{SAT} \in \mathbf{PCP}(r(n), 1)$ for $r(n) = o(\log n)$ then $\mathbf{P} = \mathbf{NP}$. This shows that the PCP Theorem is probably optimal up to constant factors. H455

11.9 (A simple PCP Theorem using logspace verifiers) Using the fact that a correct tableau can be verified in logspace, we saw the following exact characterization of **NP**:

$$\mathbf{NP} = \{L : \text{ there is a logspace machine } M \text{ s.t } x \in L \text{ iff } \exists y \ : \ M \text{ accepts } (x, y).\}.$$

Note that M has two-way access to y. Let L-PCP($r(n)$) be the class of languages whose membership proofs can be probabilistically checked by a logspace machine that uses $O(r(n))$ random bits but makes only one pass over the proof. (To use the terminology from above, it has 2-way access to x but 1-way access to y.) As in the PCP setting, "probabilistic checking of membership proofs" means that for $x \in L$ there is a proof y that the machine accepts with probability 1 and if not, the machine rejects with probability at least 1/2. Show that $\mathbf{NP} = \text{L-PCP}(\log n)$. Don't assume the PCP Theorem! H455 (This simple PCP Theorem is implicit in Lipton [Lip90]. The suggested proof is due to van Melkebeek.)

11.10 Suppose we define $J - PCP(r(n))$ similarly to $L - PCP(r(n))$, except the verifier is only allowed to read $O(r(n))$ successive bits in the membership proof. (It can decide which bits to read.) Then show that $J - PCP(\log n) \subseteq \mathbf{L}$.

11.11 This question explores why the reduction used to prove the Cook-Levin Theorem (Section 2.3) does not suffice to prove the hardness of approximation MAX-3SAT. Recall that for every **NP** language L, we defined a reduction f such that if a string $x \in L$ then $f(x) \in \mathsf{3SAT}$. Prove that there is a $x \notin L$ such that $f(x)$ is a 3SAT formula with m constraints having an assignment satisfying more than $m(1 - o(1))$ of them, where $o(1)$ denotes a function that tends to 0 with $|x|$. H455

11.12 Show a poly$(n, 1/\epsilon)$-time $(1+\epsilon)$-approximation algorithm for the knapsack problem. That is, show an algorithm that given $n + 1$ numbers $a_1, \ldots, a_n \in \mathbb{N}$ (each represented by at most n bits) and $k \in [n]$, finds a set $S \subseteq [n]$ with $|S| \leq k$ such that $\sum_{i \in S} a_i \geq \frac{\mathsf{opt}}{1+\epsilon}$ where

$$\mathsf{opt} = \max_{S \subseteq [n], |S| \leq k} \sum_{i \in S} a_i$$

H455

11.13 Show a polynomial-time algorithm that given a satisfiable 2CSP-instance φ (over binary alphabet) finds a satisfying assignment for φ.

11.14 Show a *deterministic* poly$(n, 2^q)$-time algorithm that given a qCSP-instance φ (over binary alphabet) with m clauses outputs an assignment satisfying $m/2^q$ of these assignment. H455

11.15 Prove that QUADEQ is **NP**-complete. H455

11.16 Consider the following problem: Given a system of linear equations in n with coefficients that are rational numbers, determine the largest subset of equations that are simultaneously satisfiable. Show that there is a constant $\rho < 1$ such that approximating the size of this subset is **NP**-hard. H455

11.17 Prove the assertion in the previous question when the equations are over GF(2) and each equation involves only 3 variables each.