Computational Complexity: A Modern Approach

Draft of a book: Dated January 2007
`Comments welcome!`

Sanjeev Arora and Boaz Barak
Princeton University
complexitybook@gmail.com

This is an Internet draft. Some chapters are more finished than others. References and attributions are very preliminary and we apologize in advance for any omissions (but hope you will nevertheless point them out to us).

Please send us bugs, typos, missing references or general comments to complexitybook@gmail.com — **Thank You!!**

Chapter 18

PCP and Hardness of Approximation

"...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, 1991 [**?**]

The **PCP** Theorem provides an interesting new characterization for **NP**, as the set of languages that have a "locally testable" membership proof. It is reminiscent of —and was motivated by— results such as **IP** =**PSPACE**. Its essence is the following:

Suppose somebody 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 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[1].

This new, "robust" notion of certificate/proof has an important consequence: it implies that many optimization problems are **NP**-hard not only to solve exactly but even to *approximate*. As mentioned in Chapter 2, the **P** versus **NP** question is practically important —as opposed to "just" philosophically important— because thousands of real-life combinatorial optimization problems are **NP**-hard. By showing that even computing approximate solutions to many of these problems is **NP**-hard, the **PCP** Theorem extends the practical importance of the theory of **NP**-completeness, as well as its philosophical significance.

This seemingly mysterious connection between the **PCP** Theorem —which concerns probabilistic checking of certificates— and the **NP**-hardness of computing approximate solutions is actually quite straightforward. All **NP**-hardness results ultimately derive from the Cook-Levin theorem (Section 2.3), which expresses accepting computations of a nondeterministic Turing Machine with satisfying assignments to a Boolean formula. Unfortunately, the standard representations of computation are quite nonrobust, meaning that they can be incorrect if even one bit is incorrect (see the quote at the start of this chapter). The **PCP** Theorem, by giving a *robust* representation of the certificate for **NP** languages, allow new types of reductions; see Section 18.2.3.

Below, we use the term "**PCP** Theorems" for the body of other results of a similar nature to the **PCP** Theorem that found numerous applications in complexity theory. Some important ones appear in the next Chapter, including one that improves the **PCP** Theorem so that verification is possible by reading only 3 bits in the proof!

18.1 PCP and Locally Testable Proofs

According to our usual definition, 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 $x \in L$. This means,

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

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

The class **PCP** (short for "Probabilistically Checkable Proofs") 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 $\pi[i]$.[2] (This is reminiscent of oracle TMs introduced in Chapter 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.

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

[2] Though widely used, the term "random access" is misleading since it doesn't involve any notion of randomness per se. "Indexed access" would be more accurate.

Verifiers can be *adaptive* or *nonadaptive*. A nonadaptive verifier selects its queries based only on its input and random tape, whereas an adaptive verifier can in addition 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 3 explores the power of adaptive queries.)

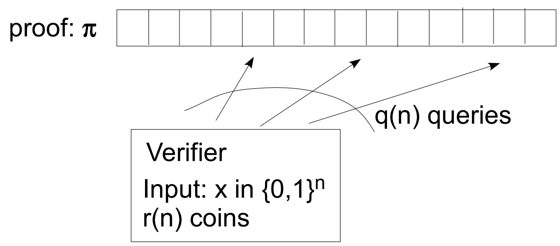

Figure 18.1: A **PCP** verifier for a language L gets an input x and random access to a string π. If $x \in L$ then there exists a string π that makes the verifier accepts, while if $x \notin L$ then the verifier rejects *every* proof π with probability at least $1/2$.

Definition 18.1 ((r,q)-verifier)
Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-*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 18.1). Then it outputs "1"(for "accept") or "0" (for "reject"). We use the notation $V^\pi(x)$ to 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 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 L has a $(c \cdot r(n), d \cdot q(n))$-verifier for some constants c, d.

Sometimes we consider verifiers for which the probability "1/2" is replaced by some other number, called the *soundness parameter*.

Theorem 18.2 (**PCP** Theorem [**?**, **?**])
$\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.

Notes:

1. Without loss of generality, proofs checkable by an (r, q)-verifier contain at most $q2^r$ bits. The verifier looks at only q places of the proof for any particular choice of its random coins, and there are only 2^r such choices. Any bit in the proof that is read with 0 probability (i.e., for no choice of the random coins) can just be deleted.

2. The previous remark implies $\mathbf{PCP}(r(n), q(n)) \subseteq \mathbf{NTIME}(2^{O(r(n))}q(n))$. The proofs checkable by an $(r(n), q(n)$-verifier have size at most $2^{O(r(n))}q(n)$. 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.

3. The constant $1/2$ in the soundness requirement of Definition 18.1 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).

Example 18.3
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 5) 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 a "scaled-up" version of the **PCP** Theorem which we will not prove.

Theorem 18.4 (Scaled-up **PCP**, [**?**, **?**, **?**])
$\mathbf{PCP}(poly, 1) = \mathbf{NEXP}$

18.2 PCP and Hardness of Approximation

The **PCP** Theorem implies that for many **NP** optimization problems, computing near-optimal solutions is no easier than computing exact solutions.

We illustrate the notion of approximation algorithms with an example. MAX 3SAT is 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.

Definition 18.5
For every 3CNF formula φ, define $\mathsf{val}(\varphi)$ to be the maximum fraction of clauses that can be satisfied by any assignment to φ's variables. In particular, if φ is satisfiable then $\mathsf{val}(\varphi) = 1$.

Let $\rho \leq 1$. 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.

In many practical settings, obtaining an approximate solution to a problem may be almost as good as solving it exactly. Moreover, for some computational problems, approximation is much easier than an exact solution.

Example 18.6 ($1/2$-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 references). (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, and by linearity of expectation, 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.)

For a few problems, one can even design $(1 - \epsilon)$-approximation algorithms for *every* $\epsilon > 0$. Exercise 10 asks you to show this for the **NP**-complete *knapsack* problem.

Researchers are extremely interested in finding the best possible approximation algorithms for **NP**-hard optimization problems. Yet, until the early 1990's most such questions were wide open. In particular, we did not know whether MAX 3SAT has a polynomial-time ρ-approximation algorithm for *every* $\rho < 1$. The **PCP** Theorem has the following Corollary.

Corollary 18.7
There exists some constant $\rho < 1$ such that if there is a polynomial-time ρ-approximation algorithm for MAX 3SAT *then* $\mathbf{P} = \mathbf{NP}$.

Later, in Chapter 19, 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 18.6 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.

18.2.1 Gap-producing reductions

To prove Corollary 18.7 for some fixed $\rho < 1$, it suffices to give a polynomial-time reduction f that maps $3CNF$ formulae to $3CNF$ formulae such that:

$$\varphi \in \mathsf{3SAT} \Rightarrow \mathsf{val}(f(\varphi)) = 1 \quad (1)$$

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

After all, if a ρ-approximation algorithm were to exist for MAX 3SAT, then we could use it to decide membership of any given formula φ in 3SAT by applying reduction f on φ and then running the approximation algorithm on the resultant $3CNF$ formula $f(\varphi)$. If $\mathsf{val}(f(\varphi) = 1$, then the approximation algorithm would return an assignment that satisfies at least ρ fraction of the clauses, which by property (2) tells us that $\varphi \in \mathsf{3SAT}$.

Later (in Section 18.2) we show that the **PCP** Theorem is equivalent to the following Theorem:

THEOREM 18.8
There exists some $\rho < 1$ and a polynomial-time reduction f satisfying (1) and (2).

By the discussion above, Theorem 18.8 implies Corollary 18.7 and so rules out a polynomial-time ρ-approximation algorithm for MAX 3SAT (unless $\mathbf{P} = \mathbf{NP}$).

Why doesn't the Cook-Levin reduction suffice to prove Theorem 18.8? The first thing one would try 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): we can always satisfy almost all of the clauses in the formulae produced by the reduction (see Exercise 9 and also the "non-robustness" quote at the start of this chapter).

18.2.2 Gap problems

The above discussion motivates the definition of *gap problems*, a notion implicit in (1) and (2). It is also an important concept in the proof of the **PCP** Theorem itself.

DEFINITION 18.9 (GAP 3SAT)
Let $\rho \in (0,1)$. The ρ-GAP 3SAT problem is to determine, given a 3CNF formula φ whether:

- φ is satisfiable, in which case we say φ is a YES instance of ρ-GAP 3SAT.
- $\mathsf{val}(\varphi) \leq \rho$, in which case we say φ is a NO instance of ρ-GAP 3SAT.

An algorithm A is said to solve ρ-$\mathsf{GAP\,3SAT}$ if $A(\varphi)=1$ if φ is a YES instance of ρ-$\mathsf{GAP\,3SAT}$ and $A(\varphi)=0$ if φ is a NO instance. Note that we do not make any requirement on $A(\varphi)$ if φ is neither a YES nor a NO instance of ρ-$\mathsf{GAP}\,q\mathsf{CSP}$.

Our earlier discussion of the desired reduction f can be formalized as follows.

Definition 18.10
Let $\rho\in(0,1)$. We say that ρ-$\mathsf{GAP\,3SAT}$ is **NP**-*hard* if for every language L there is a polynomial-time computable function f such that

$$x\in L \Rightarrow f(x) \text{ is a YES instance of } \rho\text{-}\mathsf{GAP\,3SAT}$$
$$x\notin L \Rightarrow f(x) \text{ is a NO instance of } \rho\text{-}\mathsf{GAP\,3SAT}$$

18.2.3 Constraint Satisfaction Problems

Now we generalize the definition of $\mathsf{3SAT}$ to *constraint satisfaction problems* (CSP), which allow clauses of arbitrary form (instead of just OR of literals) including those depending upon more than 3 variables. Sometimes the variables are allowed to be non-Boolean. CSPs arise in a variety of application domains and play an important role in the proof of the **PCP** Theorem.

Definition 18.11
Let q, W be natural numbers. A $q\mathsf{CSP}_W$ instance φ is a collection of functions $\varphi_1,\ldots,\varphi_m$ (called *constraints*) from $\{0..W{-}1\}^n$ 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 $f:\{0..W{-}1\}^q\to\{0,1\}$ such that $\varphi_i(\mathbf{u})=f(u_{j_1},\ldots,u_{j_q})$ for every $\mathbf{u}\in\{0..W{-}1\}^n$.

We say that an *assignment* $\mathbf{u}\in\{0..W{-}1\}^n$ *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 $\mathbf{u}\in\{0..W{-}1\}^n$. We say that φ is *satisfiable* if $\mathsf{val}(\varphi)=1$.

We call q the *arity* of φ and W the *alphabet size*. If $W=2$ we say that φ uses a *binary* alphabet and call φ a $q\mathsf{CSP}$-instance (dropping the subscript 2).

Example 18.12
$\mathsf{3SAT}$ is the subcase of $q\mathsf{CSP}_W$ where $q=3$, $W=2$, and the constraints are OR's of the involved literals.

Similarly, the **NP**-complete problem $\mathsf{3COL}$ can be viewed as a subcase of $\mathsf{2CSP}_3$ instances where for each edge (i,j), there is a constraint on the variables u_i, u_j that is satisfied iff $u_i\neq u_j$. The graph is 3-colorable iff there is a way to assign a number in $\{0,1,2\}$ to each variable such that all constraints are satisfied.

Notes:

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

2. As in the case of 3SAT, we can define maximization and gap problems for CSP instances. In particular, for any $\rho \in (0, 1)$, we define ρ-GAP qCSP$_W$ as the problem of distinguishing between a qCSP$_W$-instance φ that is satisfiable (called a YES instance) and an instance φ with $\mathsf{val}(\varphi) \leq \rho$ (called a NO instance). As before, we will drop the subscript W in the case of a binary alphabet.

3. 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. That is, for any qCSP$_W$ instance φ with m constraints, the algorithm will output an assignment satisfying $\frac{\mathsf{val}(\varphi)}{W^q}m$ constraints. Thus, unless **NP** $\subseteq$ **P**, the problem 2^{-q}-GAP qCSP is not **NP** hard.

18.2.4 An Alternative Formulation of the PCP Theorem

We now show how the **PCP** Theorem is equivalent to the **NP**-hardness of a certain gap version of qCSP. Later, we will refer to this equivalence as the "hardness of approximation viewpoint" of the **PCP** Theorem.

Theorem 18.13 (**PCP** Theorem, alternative formulation)
*There exist constants $q \in \mathbb{N}$, $\rho \in (0, 1)$ such that ρ-GAP qCSP is **NP**-hard.*

We now show Theorem 18.13 is indeed equivalent to the **PCP** Theorem:

Theorem 18.2 implies Theorem 18.13. Assume that **NP** $\subseteq$ **PCP**$(\log n, 1)$. We will show that $1/2$-GAP qCSP is **NP**-hard for some constant q. It is enough to reduce a single **NP**-complete language such as 3SAT to $1/2$-GAP qCSP 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 18.13 implies Theorem 18.2. Suppose that ρ-GAP qCSP 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). ■

Remark 18.14
Since $3CNF$ formulas are a special case of 3CSP instances, Theorem 18.8 (ρ-GAP 3SAT is **NP**-hard) implies Theorem 18.13 (ρ-GAP qCSP is **NP**-hard). Below we show Theorem 18.8 is also implied by Theorem 18.13, concluding that it is also equivalent to the **PCP** Theorem.

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

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	$\longleftrightarrow$	Maximum of $\mathsf{val}(\varphi)$ for a NO instance
Theorem 18.2 ($\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$)	$\longleftrightarrow$	Theorem 18.13 (ρ-GAP qCSP is **NP**-hard)

18.2.5 Hardness of Approximation for 3SAT and INDSET.

The CSP problem allows arbitrary functions to serve as constraints, which may seem somewhat artificial. We now show how Theorem 18.13 implies hardness of approximation results for the more natural problems of MAX 3SAT (determining the maximum number of clauses satisfiable in a 3SAT formula) and MAX INDSET (determining the size of the largest independent set in a given graph).

The following two lemmas use the **PCP** Theorem to show that unless **P** = **NP**, both MAX 3SAT and MAX INDSET are hard to approximate within a factor that is a constantless than 1. (Section 18.3 proves an even stronger hardness of approximation result for INDSET.)

Lemma 18.15 (Theorem 18.8, restated)
*There exists a constant $0 < \rho < 1$ such that ρ-GAP 3SAT is **NP**-hard.*

Lemma 18.16
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 of Lemma 18.15: Let $\epsilon > 0$ and $q \in \mathbb{N}$ be such that by Theorem 18.13, $(1-\epsilon)$-GAP qCSP is **NP**-hard. We show a reduction from $(1-\epsilon)$-GAP qCSP to $(1-\epsilon')$-GAP 3SAT where $\epsilon' > 0$ is some constant depending on ϵ and q. That is, we will show a polynomial-time function mapping YES instances of $(1-\epsilon)$-GAP qCSP to YES instances of $(1-\epsilon')$-GAP 3SAT and NO instances of $(1-\epsilon)$-GAP qCSP to NO instances of $(1-\epsilon')$-GAP 3SAT.

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 $(1-\epsilon)$-GAP qCSP (i.e., it is satisfiable) then there exists an assignment satisfying all the clauses of φ'. if φ is a NO instance of $(1-\epsilon)$-GAP qCSP 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 on $u_1, \ldots, u_q$ 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 satisfies 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 was satisfiable then so will be φ' and hence 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''. ■

Proof of Lemma 18.16: Let φ be a 3CNF formula on n variables with m clauses. We define a graph G of $7m$ vertices as follows: we associate a cluster of 7 vertices in G with each clause of φ. The vertices in cluster associated with a clause C correspond to the 7 possible assignments to the three variables C depends on (we call these *partial assignments*, since they only give values for some of the variables). For example, if C is $\overline{u_2} \vee \overline{u_5} \vee \overline{u_7}$ then the 7 vertices in the cluster associated with C correspond to all partial assignments of the form $u_1 = a, u_2 = b, u_3 = c$ for a binary vector $\langle a, b, c\rangle \neq \langle 1, 1, 1\rangle$. (If C depends on less than three variables we treat one of them as repeated and then some of the 7 vertices will correspond to the same assignment.) We put an edge between two vertices of G if they correspond to *inconsistent* partial assignments. Two partial assignments are consistent if they give the same value to all the variables they share. For example, the assignment $u_1 = 1, u_2 = 0, u_3 = 0$ is inconsistent with the assignment $u_3 = 1, u_5 = 0, u_7 = 1$ because they share a variable (u_3) to which they give a different value. In addition, we put edges between every two vertices that are in the same cluster.

Clearly transforming φ into G can be done in polynomial time. Denote by $\alpha(G)$ to be the size of the largest independent set in G. We claim that $\alpha(G) = \mathsf{val}(\varphi)m$. For starters, note that $\alpha(G) \geq \mathsf{val}(\varphi)m$. Indeed, let $\mathbf{u}$ be the assignment that satisfies $\mathsf{val}(\varphi)m$ clauses. Define a set S as follows: for each clause C satisfied by $\mathbf{u}$, put in S the vertex in the cluster associated with C that corresponds to the restriction of $\mathbf{u}$ to the variables C depends on. Because we only choose vertices that correspond to restrictions of the assignment $\mathbf{u}$, no two vertices of S correspond to inconsistent assignments and hence S is an independent set of size $\mathsf{val}(\varphi)m$.

Suppose that G has an independent set S of size k. We will use S to construct an assignment $\mathbf{u}$ satisfying k clauses of φ, thus showing that $\mathsf{val}(\varphi)m \geq \alpha(G)$. We define $\mathbf{u}$ as follows: for every $i \in [n]$, if there is a vertex in S whose partial assignment gives a value a to u_i, then set $u_i = a$; otherwise set $u_i = 0$. This is well defined because S is an independent set, and each variable u_i can get at most a single value by assignments corresponding to vertices in S. On the other hand, because we put all the edges within each cluster, S can contain at most a single vertex in each cluster, and hence there are k distinct cluster with members in S. By our definition of $\mathbf{u}$ it satisfies all the clauses associated with these clusters. ■

Remark 18.17
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* to the fact that $x \in L$ to a certificate to the fact that $x' \in L'$. Although the definition of a Karp reduction does not require that this mapping is efficient, it often turned out that the proof did provide a way to compute this mapping in polynomial time. The way we proved that $f(x) \in L' \Rightarrow x \in L$ was by showing a way to map a certificate to the fact that $x' \in L'$ to a certificate to the fact that $x \in L$. Once again, the proofs typically yield an efficient way to compute this mapping.

A similar thing happens in the gap preserving reductions used in the proofs of Lemmas 18.15 and 18.16 and elsewhere in this chapter. When reducing from, say, ρ-GAP qCSP to ρ'-GAP 3SAT we show a function f that maps a CSP instance φ to a 3SAT instance ψ satisfying the following two properties:

Completeness We can map a satisfying assignment of φ to a satisfying assignment to ψ

Soundness Given any assignment that satisfies more than a ρ' fraction of ψ's clauses, we can map it back into an assignment satisfying more than a ρ fraction of φ's constraints.

18.3 $n^{-\delta}$-approximation of independent set is NP-hard.

We now show a much stronger hardness of approximation result for the independent set (INDSET) problem than Lemma 18.16. Namely, we show that there exists a constant $\delta \in (0,1)$ such that unless $\mathbf{P} = \mathbf{NP}$, there is no polynomial-time n^{δ}-approximation algorithm for INDSET. That is, we show that if there is a polynomial-time algorithm A that given an n-vertex graph G outputs an independent set of size at least $\frac{\mathsf{opt}}{n^{\delta}}$ (where opt is the size of the largest independent set in G) then $\mathbf{P} = \mathbf{NP}$. We note that an even stronger result is known: the constant δ can be made arbitrarily close to 1 [**?**, **?**]. This factor is almost optimal since the independent set problem has a trivial n-approximation algorithm: output a single vertex.

Our main tool will be the notion of *expander graphs* (see Note 18.18 and Chapter **??**). Expander graphs will also be used in the proof of **PCP** Theorem itself. We use here the following property of expanders:

Lemma 18.19
Let $G = (V, E)$ be a λ-expander graph for some $\lambda \in (0,1)$. Let S be a subset of V with $|S| = \beta|V|$ for some $\beta \in (0,1)$. Let $(X_1, \ldots, X_\ell)$ be a tuple of random variables denoting the vertices of a uniformly chosen $(\ell$–$1)$-step path in G. Then,

$$(\beta - 2\lambda)^k \leq \Pr[\forall_{i\in[\ell]} X_i \in S] \leq (\beta + 2\lambda)^k$$

The upper bound of Lemma 18.19 is implied by Theorem **??**; we omit the proof of the lower bound.

The hardness result for independent set follows by combining the following lemma with Lemma 18.16:

NOTE 18.18 (EXPANDER GRAPHS)
Expander graphs are described in Chapter **??**. We define there a parameter $\lambda(G) \in [0,1]$, for every regular graph G (see Definition 7.25). The main property we need in this chapter is that for every regular graph $G = (V, E)$ and every $S \subseteq V$ with $|S| \leq |V|/2$,

$$\Pr_{(u,v)\in E}[u \in S, v \in S] \leq \frac{|S|}{|V|}\left(\frac{1}{2} + \frac{\lambda(G)}{2}\right) \quad (3)$$

Another property we use is that $\lambda(G^\ell) = \lambda(G)^\ell$ for every $\ell \in \mathbb{N}$, where G^ℓ is obtained by taking the adjacency matrix of G to the ℓ^{th} power (i.e., an edge in G^ℓ corresponds to an $(\ell$–$1)$-step path in G).
For every $c \in (0,1)$, we call a regular graph G satisfying $\lambda(G) \leq c$ a c-*expander graph*. If $c < 0.9$, we drop the prefix c and simply call G an *expander graph*. (The choice of the constant 0.9 is arbitrary.) As shown in Chapter **??**, for every constant $c \in (0,1)$ there is a constant d and an algorithm that given input $n \in N$, runs in poly(n) time and outputs the adjacency matrix of an n-vertex d-regular c-expander (see Theorem 16.32).

LEMMA 18.20
For every $\lambda > 0$ there is a polynomial-time computable reduction f that maps every n-vertex graph G into an m-vertex graph H such that

$$(\tilde{\alpha}(G) - 2\lambda)^{\log n} \leq \tilde{\alpha}(H) \leq (\tilde{\alpha}(G) + 2\lambda)^{\log n}$$

where $\tilde{\alpha}(G)$ is equal to the fractional size of the largest independent set in G.

Recall that Lemma 18.16 shows that there are some constants $\beta, \epsilon \in (0,1)$ such that it is **NP**-hard to tell whether a given graph G satisfies **(1)** $\tilde{\alpha}(G) \geq \beta$ or **(2)** $\tilde{\alpha}(G) \leq (1-\epsilon)\beta$. By applying to G the reduction of Lemma 18.20 with parameter $\lambda = \beta\epsilon/8$ we get that in case **(1)**, $\tilde{\alpha}(H) \geq (\beta - \beta\epsilon/4)^{\log n} = (\beta(1-\epsilon/4))^{\log n}$, and in case **(2)**, $\tilde{\alpha}(H) \leq ((1-\epsilon)\beta + \beta\epsilon/4)^{\log n} = (\beta(1-0.75\epsilon))^{\log n}$. We get that the gap between the two cases is equal to $c^{\log n}$ for some $c > 1$ which is equal to m^δ for some $\delta > 0$ (where $m = \text{poly}(n)$ is the number of vertices in H).

PROOF OF LEMMA 18.20: Let G, λ be as in the lemma's statement. We let K be an n-vertex λ-expander of degree d (we can obtain such a graph in polynomial-time, see Note 18.18). We will map G into a graph H of $nd^{\log n - 1}$ vertices in the following way:

- The vertices of H correspond to all the $(\log n$–$1)$-step paths in the λ-expander K.
- We put an edge between two vertices u, v of H corresponding to the paths $\langle u_1, \ldots, u_{\log n}\rangle$ and $\langle v_1, \ldots, v_{\log n}\rangle$ if there exists an edge in G between two vertices in the set $\{u_1, \ldots, u_{\log n}, v_1, \ldots, v_{\log n}\}$.

A subset T of H's vertices corresponds to a subset of $\log n$-tuples of numbers in $[n]$, which we can identify as tuples of vertices in G. We let $V(T)$ denote the set of all the vertices appearing in one of the tuples of T. Note that in this notation, T is an independent set in H if and only if $V(T)$ is an independent set of G. Thus for every independent set T in H, we have that $|V(T)| \leq \tilde{\alpha}(G)n$ and hence by the upper bound of Lemma 18.19, T takes up less than an $(\tilde{\alpha}(H) + 2\lambda)^{\log n}$ fraction of H's vertices. On the other hand, if we let S be the independent set of G of size $\tilde{\alpha}(G)n$ then by the lower bound of Lemma 18.19, an $(\tilde{\alpha} - 2\lambda)^{\log n}$ fraction of H's vertices correspond to paths fully contained in S, implying that $\tilde{\alpha}(H) \geq (\tilde{\alpha}(G) - 2\lambda)^{\log n}$. ■

18.4 $\mathbf{NP} \subseteq \mathbf{PCP}(\mathbf{poly}(n), 1)$: PCP based upon 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, this section builds up to a stronger Corollary 18.26, which will be used in the proof of the **PCP** theorem.

Theorem 18.21
$\mathbf{NP} \subseteq \mathbf{PCP}(\mathit{poly}(n), 1)$

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.

18.4.1 Tool: Linearity Testing and the Walsh-Hadamard Code

We use the *Walsh-Hadamard code* (see Section 17.5, 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); namely, the function $\mathsf{WH} : \{0,1\}^* \to \{0,1\}^*$ mapping 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 $\{0,1\}$.

The Walsh-Hadamard code is an error correcting code with minimum distance $1/2$, by which we mean that for every $\mathbf{u} \neq \mathbf{u}' \in \{0,1\}^n$, the encodings $\mathsf{WH}(\mathbf{u})$ and $\mathsf{WH}(\mathbf{u})$ differ in half the bits. This follows from the familiar random subsum principle (Claim A.5) since exactly half of the strings $\mathbf{x} \in \{0,1\}^n$ satisfy $\mathbf{u} \odot \mathbf{x} \neq \mathbf{u}' \odot \mathbf{x}$. 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

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}) \tag{4}$$

for all the 2^{2n} pairs $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ (where "+" on the left side of (pcp:eq:lintest) denotes vector addition over $\mathrm{GF}(2)^n$ and on the right side denotes addition over $\mathrm{GF}(2)$).

But can we test f by querying it in only a *constant* number of places? Clearly, if f is not linear but very close to being a linear function (e.g., if f is obtained by modifying a linear function on a very small fraction of its inputs) then such a *local* test will not be able to distinguish f from a linear function. Thus we set our goal on 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*. It turns out that the natural test of choosing $\mathbf{x}, \mathbf{y}$ at random and verifying (4) achieves this goal:

Definition 18.22
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 18.23 (Linearity Testing [**?**])
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 18.23 to Section 19.3 of the next chapter. 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 $(1\text{–}\delta)$-close to a linear function, by testing Condition (4) repeatedly $O(1/\delta)$ times with independent randomness. We call such a test a *$(1\text{–}\delta)$-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 $(1\text{–}\delta)$-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 $\tilde{f}(\mathbf{x}) = \mathbf{y}' + \mathbf{y}''$.[3] 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.

18.4.2 Proof of Theorem 18.21

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 $\text{GF}(2) = \{0, 1\}$ that are satisfiable.

Example 18.24
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.

More generally, an instance of QUADEQ over the variables $u_1, \ldots, u_n$ is of the form $AU = b$, where U is the n^2-dimensional vector whose $\langle i, j \rangle^{th}$ entry is u_iu_j, A is an $m \times n^2$ matrix and $\mathbf{b} \in \{0, 1\}^m$. In other words, U is the *tensor product* $\mathbf{u} \otimes \mathbf{u}$, where $\mathbf{x} \otimes \mathbf{y}$ for a pair of vectors $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$ denotes the n^2-dimensional vector (or $n \times n$ matrix) whose (i, j) entry is x_iy_j. For every $i, j \in [n]$ with $i \leq j$, the entry $A_{k,\langle i,j \rangle}$ is the coefficient of u_iu_j in the k^{th} equation (we identify $[n^2]$ with $[n] \times [n]$ in some canonical way). The vector $\mathbf{b}$ consists of the right hand side of the m equations. Since $u_i = (u_i)^2$ in GF(2), we can assume the equations do not contain terms of the form u_i^2.

Thus a satisfying assignment consists of $u_1, u_2, \ldots, u_n \in \text{GF}(2)$ such that its tensor product $U = u \otimes u$ satisfies $AU = b$. We leave it as Exercise 12 to show that QUADEQ, the language of all satisfiable instances, is indeed **NP**-complete.

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 correct **PCP** proof π for A, b will consist of the Walsh-Hadamard encoding for $\mathbf{u}$ and the Walsh-Hadamard encoding for $\mathbf{u} \otimes \mathbf{u}$, by which we mean that we will design the **PCP** verifier in a way ensuring that it accepts proofs

[3] We use here the fact that over GF(2), $a + b = a - b$.

Figure 18.2: The **PCP** proof that a set of quadratic equations is satisfiable consists of $\mathsf{WH}(u)$ and $\mathsf{WH}(u \otimes u)$ 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.

of this form with probability one, satisfying the completeness condition. (Note that π is of length $2^n + 2^{n^2}$.)

Below, we repeatedly use the following fact:

RANDOM SUBSUM PRINCIPLE: *If $\mathbf{u} \neq \mathbf{v}$ then for at least $1/2$ the choices of $\mathbf{x}$, $\mathbf{u} \odot \mathbf{x} \neq \mathbf{v} \odot \mathbf{x}$. Realize that $\mathbf{x}$ can be viewed as a random subset of indices in $[1, \ldots, n]$ and the principle says that with probability $1/2$ the sum of the u_i's over this index set is different from the corresponding sum of v_i's.*

The verifier. 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 $g : \{0,1\}^{n^2} \to \{0,1\}$.

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 0.99-linearity test on both f, g, and rejects the proof at once if either test fails.

Thus, if either of f, g is not 0.99-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 0.99-close to f, and $\tilde{g}$ is 0.99-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 15) number 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 in the rest of the procedure we use f, g for $\tilde{f}, \tilde{g}$ respectively. Furthermore, 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.

Verifier V does the following test 3 times: "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, $w = \mathbf{u} \otimes \mathbf{u}$, 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 three 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 $1 - (3/4)^3 = 37/64$.) 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 test 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,

$$\sum_{i,j} A_{k,(i,j)} u_i u_j = b_k. \tag{5}$$

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 (5) 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.

(Actually, the above test has to be repeated twice to ensure that if $\mathbf{u}$ does not satisfy the system, then Step 3 rejects with probability at least $3/4$.)

18.4.3 PCP's of proximity

Theorem 18.21 says that (exponential-sized) certificates for **NP** languages can be checked by examining only $O(1)$ bits in them. The proof actually yields a somewhat stronger result, which will be used in the proof of the **PCP** Theorem. This concerns the following scenario: we hold a circuit C in our hands that has n input wires. Somebody holds a satisfying assignment u. He writes down $\mathsf{WH}(u)$ as well as another string π for us. We do a probabilistic test on this by examining $O(1)$ bits in these strings, and at the end we are convinced of this fact.

Concatenation test. First we need to point out a property of Walsh-Hadamard codes and a related *concatenation test*. In this setting, we are given two linear functions f, g that encode strings of lengths n and $n+m$ respectively. We have to check by examining only $O(1)$ bits in f, g that if $\mathbf{u}$ and $\mathbf{v}$ are the strings encoded by f, g (that is, $f = \mathsf{WH}(\mathbf{u})$ and $h = \mathsf{WH}(\mathbf{v})$) then $\mathbf{u}$ is the same as the first n bits of $\mathbf{v}$. By the random subsum principle, the following simple test rejects with probability $1/2$ if this is not the case. Pick a random $\mathbf{x} \in \{0,1\}^n$, and denote by $\mathbf{X} \in \mathrm{GF}(2)^{m+n}$ the string whose first n bits are $\mathbf{x}$ and the remaining bits are all-0. Verify that $f(X) = g(\mathbf{x})$.

With this test in hand, we can prove the following corollary.

Corollary 18.25 (Exponential-sized **PCP** of proximity.)
There exists a verifier V that given any circuit C of size m and with n inputs has the following property:

1. *If $\mathbf{u} \in \{0,1\}^n$ is a satisfying assignment for circuit C, then there is a string π_2 of size $2^{\mathrm{poly}(m)}$ such that V accepts $\mathsf{WH}(\mathbf{u}) \circ \pi_2$ with probability 1. (Here $\circ$ denotes concatenation.)*
2. *For every strings $\pi_1, \pi_2 \in \{0,1\}^*$, where π_1 has 2^n bits, if V accepts $\pi_1 \circ \pi_2$ with probability at least $1/2$, then π_1 is 0.99-close to $\mathsf{WH}(\mathbf{u})$ for some $\mathbf{u}$ that satisfies C.*
3. *V uses poly(m) random bits and examines only $O(1)$ bits in the provided strings.*

Proof: One looks at the proof of **NP**-completeness of QUADEQ to realize that given circuit C with n input wires and size m, it yields an instance of QUADEQ of size $O(m)$ such that $\mathbf{u} \in \{0,1\}^n$ satisfies the circuit iff there is a string $\mathbf{v}$ of size $M = O(m)$ such that $\mathbf{u} \circ \mathbf{v}$ satisfies the instance of QUADEQ. (Note that we are thinking of $\mathbf{u}$ both as a string of bits that is an input to C and as a string over $\mathrm{GF}(2)^n$ that is a partial assignment to the variables in the instance of QUADEQ.)

The verifier expects π_2 to contain whatever our verifier of Theorem 18.21 expects in the proof for this instance of QUADEQ, namely, a linear function f that is $\mathsf{WH}(w)$, and another linear function g that is $\mathsf{WH}(w \otimes w)$ where w satisfies the QUADEQ instance. The verifier checks these functions as described in the proof of Theorem 18.21.

However, in the current setting our verifer is also given a string $\pi_1 \in \{0,1\}^{2^n}$. Think of this as a function $h : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$. The verifier checks that h is 0.99-close to a linear function, say $\tilde{h}$. Then to check that $\tilde{f}$ encodes a string whose first n bits are the same as the string encoded by $\tilde{h}$, the verifier does a concatenation test.

Clearly, the verifier only reads $O(1)$ bits overall. ■

The following Corollary is also similarly proven and is the one that will actually be used later. It concerns a similar situation as above, except the inputs to the circuit C are thought of as the concatenation of two strings of lengths n_1, n_2 respectively where $n = n_1 + n_2$.

Corollary 18.26 (**PCP** of proximity when assignment is in two pieces)
There exists a verifier V that given any circuit C with n input wires and size m and also two numbers n_1, n_2 such that $n_1 + n_2 = n$ has the following property:

1. *If $\mathbf{u_1} \in \{0,1\}^{n_1}, \mathbf{u_2} \in \{0,1\}^{n_2}$ is such that $\mathbf{u_1} \circ \mathbf{u_2}$ is a satisfying assignment for circuit C, then there is a string π_3 of size $2^{\mathrm{poly}(m)}$ such that V accepts $\mathsf{WH}(\mathbf{u_1}) \circ \mathsf{WH}(\mathbf{u_2}) \circ \pi_3$ with probability 1.*

2. *For every strings $\pi_1, \pi_2, \pi_3 \in \{0,1\}^*$, where π_1 and π_2 have 2^{n_1} and 2^{n_2} bits respectively, if V accepts $\pi_1 \circ \pi_2 \circ \pi_3$ with probability at least $1/2$, then π_1, π_2 are 0.99-close to $\mathsf{WH}(\mathbf{u_1})$, $\mathsf{WH}(\mathbf{u_2})$ respectively for some $\mathbf{u_1}, \mathbf{u_2}$ such that $\mathbf{u_1} \circ \mathbf{u_2}$ is a satisfying assignment for circuit C.*

3. *V uses poly(m) random bits and examines only $O(1)$ bits in the provided strings.*

18.5 Proof of the PCP Theorem.

As we have seen, the **PCP** Theorem is equivalent to Theorem 18.13, stating that ρ-$\mathsf{GAP}\,q\mathsf{CSP}$ is **NP**-hard for some constants q and $\rho < 1$. Consider the case that $\rho = 1-\epsilon$ where ϵ is not necessarily a constant but can be a function of m (the number of constraints). Since the number of satisfied constraints is always a whole number, if φ is unsatisfiable then $\mathsf{val}(\varphi) \leq 1 - 1/m$. Hence, the gap problem $(1{-}1/m)$-$\mathsf{GAP}\,3\mathsf{CSP}$ is a generalization of $3\mathsf{SAT}$ and is **NP** hard. The idea behind the proof is to start with this observation, and iteratively show that $(1{-}\epsilon)$-$\mathsf{GAP}\,q\mathsf{CSP}$ is **NP**-hard for larger and larger values of ϵ, until ϵ is as large as some absolute constant independent of m. This is formalized in the following lemma.

Definition 18.27
Let f be a function mapping CSP instances to CSP instances. We say that f is a *CL-reduction* (short for *complete linear-blowup reduction*) if it is polynomial-time computable and for every CSP instance φ with m constraints, satisfies:

Completeness: If φ is satisfiable then so is $f(\varphi)$.

Linear blowup: The new $q\mathsf{CSP}$ instance $f(\varphi)$ has at most Cm constraints and alphabet W, where C and W can depend on the arity and the alphabet size of φ (but not on the number of constraints or variables).

Lemma 18.28 (**PCP** Main Lemma)
There exist constants $q_0 \geq 3$, $\epsilon_0 > 0$, and a CL-reduction f such that for every $q_0\mathsf{CSP}$-instance φ with binary alphabet, and every $\epsilon < \epsilon_0$, the instance $\psi = f(\varphi)$ is a $q_0\mathsf{CSP}$ (over binary alphabet) satisfying

$$\mathsf{val}(\varphi) \leq 1 - \epsilon \Rightarrow \mathsf{val}(\psi) \leq 1 - 2\epsilon$$

Lemma 18.28 can be succinctly described as follows:

	Arity	Alphabet	Constraints	Value
Original	q_0	binary	m	$1-\epsilon$
	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
Lemma 18.28	q_0	binary	Cm	$1-2\epsilon$

This Lemma allows us to easily prove the **PCP** Theorem.

Proving Theorem 18.2 from Lemma 18.28. Let $q_0 \geq 3$ be as stated in Lemma 18.28. As already observed, the decision problem q_0CSP is **NP**-hard. To prove the **PCP** Theorem we give a reduction from this problem to GAP q_0CSP. Let φ be a q_0CSP instance. Let m be the number of constraints in φ. If φ is satisfiable then $\mathsf{val}(\varphi) = 1$ and otherwise $\mathsf{val}(\varphi) \leq 1 - 1/m$. We use Lemma 18.28 to amplify this gap. Specifically, apply the function f obtained by Lemma 18.28 to φ a total of $\log m$ times. We get an instance ψ such that if φ is satisfiable then so is ψ, but if φ is not satisfiable (and so $\mathsf{val}(\varphi) \leq 1 - 1/m$) then $\mathsf{val}(\psi) \leq 1 - \min\{2\epsilon_0, 1 - 2^{\log m}/m\} = 1 - 2\epsilon_0$. Note that the size of ψ is at most $C^{\log m} m$, which is polynomial in m. Thus we have obtained a gap-preserving reduction from L to the $(1{-}2\epsilon_0)$-GAP q_0CSP problem, and the **PCP** theorem is proved. ■

The rest of this section proves Lemma 18.28 by combining two transformations: the first transformation amplifies the gap (i.e., fraction of violated constraints) of a given CSP instance, at the expense of increasing the alphabet size. The second transformation reduces back the alphabet to binary, at the expense of a modest reduction in the gap. The transformations are described in the next two lemmas.

Lemma 18.29 (Gap Amplification [?])
For every $\ell \in \mathbb{N}$, there exists a CL-reduction g_ℓ such that for every CSP *instance φ with binary alphabet, the instance $\psi = g_\ell(\varphi)$ has has arity only 2 (but over a non-binary alphabet) and satisfies:*

$$\mathsf{val}(\varphi) \leq 1 - \epsilon \Rightarrow \mathsf{val}(\psi) \leq 1 - \ell\epsilon$$

for every $\epsilon < \epsilon_0$ where $\epsilon_0 > 0$ is a number depending only on ℓ and the arity q of the original instance φ.

Lemma 18.30 (Alphabet Reduction)
There exists a constant q_0 and a CL- reduction h such that for every CSP *instance φ, if φ had arity two over a (possibly non-binary) alphabet $\{0..W{-}1\}$ then $\psi = h(\varphi)$ has arity q_0 over a* binary *alphabet and satisfies:*

$$\mathsf{val}(\varphi) \leq 1 - \epsilon \Rightarrow \mathsf{val}(h(\varphi)) \leq 1 - \epsilon/3$$

Lemmas 18.29 and 18.30 together imply Lemma 18.28 by setting $f(\varphi) = h(g_6(\varphi))$. Indeed, if φ was satisfiable then so will $f(\varphi)$. If $\mathsf{val}(\varphi) \leq 1 - \epsilon$, for $\epsilon < \epsilon_0$ (where ϵ_0 the value obtained in Lemma 18.29 for $\ell = 6$, $q = q_0$) then $\mathsf{val}(g_6(\varphi)) \leq 1 - 6\epsilon$ and hence $\mathsf{val}(h(g_6(\varphi))) \leq 1 - 2\epsilon$. This composition is described in the following table:

	Arity	Alphabet	Constraints	Value
Original	q_0	binary	m	$1-\epsilon$
	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
Lemma 18.29	2	W	Cm	$1-6\epsilon$
	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
Lemma 18.30	q_0	binary	$C'Cm$	$1-2\epsilon$

18.5.1 Gap Amplification: Proof of Lemma 18.29

To prove Lemma 18.29, we need to exhibit a function g that maps a $q\mathsf{CSP}$ instance to a $2\mathsf{CSP}_W$ instance over a larger alphabet $\{0..W{-}1\}$ in a way that increases the fraction of violated constraints.

We will show that we may assume without loss of generality that the instance of $q\mathsf{CSP}$ has a specific form. To describe this we need a definition.

We will assume that the instance satisfies the following properties, since we can give a simple CL-reduction from $q\mathsf{CSP}$ to this special type of $q\mathsf{CSP}$. (See the "Technical Notes" section at the end of the chapter.) We will call such instances "nice."

Property 1: The arity q is 2 (though the alphabet may be nonbinary).

Property 2: Let the *constraint graph* of ψ be the graph G with vertex set $[n]$ where for every constraint of φ depending on the variables u_i, u_j, the graph G has the edge (i, j). We allow G to have parallel edges and self-loops. Then G is d-regular for some constant d (independent of the alphabet size) and at every node, half the edges incident to it are self-loops.

Property 3: The constraint graph is an expander.

The rest of the proof consists of a "powering" operation for nice $2\mathsf{CSP}$ instances. This is described in the following Lemma.

Lemma 18.31 (Powering)
Let ψ be a $2\mathsf{CSP}_W$ instance satisfying Properties 1 through 3. For every number t, there is an instance of $2\mathsf{CSP}$ ψ^t such that:

1. *ψ^t is a $2\mathsf{CSP}_{W'}$-instance with alphabet size $W' < W^{d^{5t}}$, where d denote the degree of ψ's constraint graph. The instance ψ^t has $d^{t+\sqrt{t}}n$ constraints, where n is the number of variables in ψ.*

2. *If ψ is satisfiable then so is ψ^t.*

3. *For every $\epsilon < \frac{1}{d\sqrt{t}}$, if $\mathsf{val}(\psi) \leq 1 - \epsilon$ then $\mathsf{val}(\psi^t) \leq 1 - \epsilon'$ for $\epsilon' = \frac{\sqrt{t}}{10^5 d W^4}\epsilon$.*

4. *The formula ψ^t is computable from ψ in time polynomial in m and W^{d^t}.*

Proof: Let ψ be a $2\mathsf{CSP}_W$-instance with n variables and $m = nd$ constraints, and as before let G denote the *constraint graph* of ψ.

The formula ψ^t will have the same number of variables as ψ. The new variables $\mathbf{y} = y_1, \ldots, y_n$ take values over an alphabet of size $W' = W^{d^{5t}}$, and thus a value of a new variable y_i is a d^{5t}-tuple of values in $\{0..W{-}1\}$. We will think of this tuple as giving a value in $\{0..W{-}1\}$ to every old variable u_j where j can be reached from u_i using a path of at most $t + \sqrt{t}$ steps in G (see Figure 18.3). In other words, the tuple contains an assignment for every u_j such that j is in the ball of radius $t + \sqrt{t}$ and center i in G. For this reason, we will often think of an assignment to y_i as "claiming" a certain value for u_j. (Of course, another variable y_k could claim a different value for u_j.) Note that since G has degree d, the size of each such ball is no more than $d^{t+\sqrt{t}+1}$ and hence this information can indeed be encoded using an alphabet of size W'.

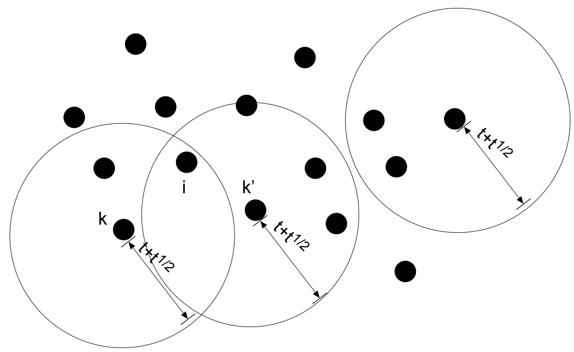

Figure 18.3: An assignment to the formula ψ^t consists of n variables over an alphabet of size less than $W^{d^{5t}}$, where each variable encodes the restriction of an assignment of ψ to the variables that are in some ball of radius $t + \sqrt{t}$ in ψ's constraint graph. Note that an assignment $y_1, \ldots, y_n$ to ψ^t may be *inconsistent* in the sense that if i falls in the intersection of two such balls centered at k and k', then y_k may claim a different value for u_i than the value claimed by $y_{k'}$.

For every $(2t+1)$-step path $p = \langle i_1, \ldots, i_{2t+2} \rangle$ in G, we have one corresponding constraint C_p in ψ^t (see Figure 18.4). The constraint C_p depends on the variables y_{i_1} and $y_{i_{2t+1}}$ and outputs FALSE if (and only if) there is some $j \in [2t+1]$ such that:

1. i_j is in the $t + \sqrt{t}$-radius ball around i_1.
2. i_{j+1} is in the $t + \sqrt{t}$-radius ball around i_{2t+2}
3. If w denotes the value y_{i_1} claims for u_{i_j} and w' denotes the value $y_{i_{2t+2}}$ claims for $u_{i_{j+1}}$, then the pair (w, w') violates the constraint in φ that depends on u_{i_j} and $u_{i_{j+1}}$.

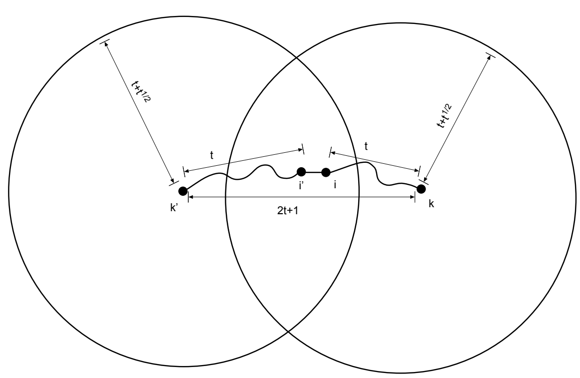

Figure 18.4: ψ^t has one constraint for every path of length $2t+1$ in ψ's constraint graph, checking that the views of the balls centered on the path's two endpoints are consistent with one another and the constraints of ψ.

A few observations are in order. First, the time to produce such an assignment is polynomial in m and W^{d^t}, so part 4 of Lemma 18.29 is trivial.

Second, for every assignment to $u_1, u_2, \ldots, u_n$ we can "lift" it to a *canonical* assignment to $y_1, \ldots, y_n$ by simply assigning to each y_i the vector of values assumed by u_j's that lie in a ball of radius $t + \sqrt{t}$ and center i in G. If the assignment to the u_j's was a satisfying assignment for ψ, then this canonical assignment satisfies ψ^t, since it will satisfy all constraints encountered in walks of length $2t + 1$ in G. Thus part 2 of Lemma 18.29 is also trivial.

This leaves part 3 of the Lemma, the most difficult part. We have to show that if $\mathsf{val}(\psi) \leq 1 - \epsilon$ then every assignment to the y_i's satisfies at most $1 - \epsilon'$ fraction of constraints in ψ^t, where $\epsilon < \frac{1}{d\sqrt{t}}$ and $\epsilon' = \frac{\sqrt{t}}{10^5 dW^4}\epsilon$. This is tricky since an assignment to the y_i's does not correspond to any obvious assignment for the u_i's: for each u_j, different values could be claimed for it by different y_i's. The intuition will be to show that these *inconsistencies* among the y_i's can't happen too often (at least if the assignment to the y_i's satisfies $1 - \epsilon'$ constraints in ψ^t).

From now on, let us fix some arbitrary assignment $\mathbf{y} = y_1, \ldots, y_n$ to ψ^t's variables. The following notion is key.

The plurality assignment: For every variable u_i of ψ, we define the random variable Z_i over $\{0, \ldots, W - 1\}$ to be the result of the following process: starting from the vertex i, take a t step random walk in G to reach a vertex k, and output the value that y_k claims for u_i. We let z_i denote the *plurality* (i.e., most likely) value of Z_i. If more than one value is most likely, we break ties arbitrarily. This assignment is called a *plurality assignment* (see Figure 18.5). Note that $Z_i = z_i$ with probability at least $1/W$.

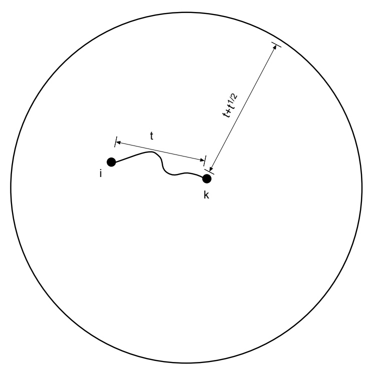

Figure 18.5: An assignment y for ψ^t induces a plurality assignment u for ψ in the following way: u_i gets the most likely value that is claimed for it by y_k, where k is obtained by taking a t-step random walk from i in the constraint graph of ψ.

Since $\mathsf{val}(\psi) \leq 1 - \epsilon$, *every* assignment for ψ fails to satisfy $1 - \epsilon$ fraction of the constraints, and this is therefore also true for the plurality assignment. Hence there exists a set F of $\epsilon m = \epsilon nd$ constraints in ψ that are violated by the assignment $\mathbf{z} = z_1, \ldots, z_n$. We will use this set F to show that at least an ϵ' fraction of ψ^t's constraints are violated by the assignment $\mathbf{y}$.

Why did we define the plurality assignment $\mathbf{z}$ in this way? The reason is illustrated by the following claim, showing that for every edge $f = (i, i')$ of G, among all paths that contain the edge f somewhere in their "midsection", most paths are such that the endpoints of the path claim the plurality values for u_i and $u_{i'}$.

Claim 18.32
For every edge $f = (i, i')$ in G define the event $B_{j,f}$ over the set of $(2t{+}1)$-step paths in G to contain all paths $\langle i_1, \ldots, i_{2t+2} \rangle$ satisfying:

- *f is the j^{th} edge in the path. That is, $f = (i_j, i_{j+1})$.*
- *y_{i_1} claims the plurality value for u_i.*
- *$y_{i_{2t+2}}$ claims the plurality value for $u_{i'}$.*

Let $\delta = \frac{1}{100W^2}$. Then for every $j \in \left\{t, \ldots, t + \delta\sqrt{t}\right\}$, $\Pr[B_{j,f}] \geq \frac{1}{nd2W^2}$.

Proof: First, note that because G is regular, the j^{th} edge of a random path is a random edge, and hence the probability that f is the j^{th} edge on the path is equal to $\frac{1}{nd}$. Thus, we need to prove that,

$$\Pr[\text{endpoints claim plurality values for } u_i, u_{i'} \text{ (resp.)} | f \text{ is } j^{th} \text{ edge}] \geq \frac{1}{2W^2} \tag{6}$$

We start with the case $j = t + 1$. In this case (6) holds essentially by definition: the left-hand side of (6) is equal to the probability that the event that the endpoints claim the plurality for these variables happens for a path obtained by joining a random t-step path from i to a random t-step path from i'. Let k be the endpoint of the first path and k' be the endpoint of the second path. Let W_i be the distribution of the value that y_k claims for u_i, where k is chosen as above, and similarly define $W_{i'}$ to be the distribution of the value that $y_{k'}$ claims for $u_{i'}$. Note that since k and k' are chosen independently, the random variables W_i and $W_{i'}$ are independent. Yet by definition the distribution of W_i identical to the distribution Z_i, while the distribution of $W_{i'}$ is identical to $Z_{i'}$. Thus,

$$\Pr[\text{endpoints claim plurality values for } u_i, u_{i'} \text{ (resp.)} | f \text{ is } j^{th} \text{ edge}] =$$
$$\Pr_{k,k'}[W_i = z_i \wedge W_{i'} = z_{i'}] = \Pr_k[W_i = z_i] \Pr_{k'}[W_{i'} = z_{i'}] \geq \tfrac{1}{W^2}$$

In the case that $j \neq 2t + 1$ we need to consider the probability of the event that endpoints claim the plurality values happening for a path obtained by joining a random $t - 1 + j$-step path from i to a random $t + 1 - j$-step path from i' (see Figure 18.6). Again we denote by k the endpoint of the first path, and by k' the endpoint of the second path, by W_i the value y_k claims for u_i and by $W_{i'}$ the value $y_{k'}$ claims for $u_{i'}$. As before, W_i and $W_{i'}$ are independent. However, this time W_i and Z_i may not be identically distributed. Fortunately, we can show that they are almost identically distributed, in other words, the distributions are *statistically close*. Specifically, because half of the constraints involving each variable are self loops, we can think of a t-step random walk from a vertex i as follows: **(1)** throw t coins and let S_t denote the number of the coins that came up "heads" **(2)** take S_t "real" (non self-loop) steps on the graph. Note that the endpoint of a t-step random walk and a t'-step random walk will be identically distributed if in Step **(1)** the variables S_t and $S_{t'}$ turn out to be the same number. Thus, the statistical distance of the endpoint of a t-step random walk and a t'-step random walk is bounded by the statistical distance of S_t and $S_{t'}$ where S_ℓ denotes the binomial distribution of the sum of ℓ balanced independent coins.

However, the distributions S_t and $S_{t+\delta\sqrt{t}}$ are within statistical distance at most 10δ for every δ, t (see Exercise 15) and hence in our case W_i and $W_{i'}$ are $\frac{1}{10W}$-close to Z_i and $Z_{i'}$ respectively. Thus $|\Pr_k[W_i = z_i] - \Pr[Z_i = z_i]| < \frac{1}{10W}$, $|\Pr_k[W_{i'} = z_{i'}] - \Pr[Z_{i'} = z_{i'}]| < \frac{1}{10W}$ which proves (6) also for the case $j \neq 2t+1$. ■

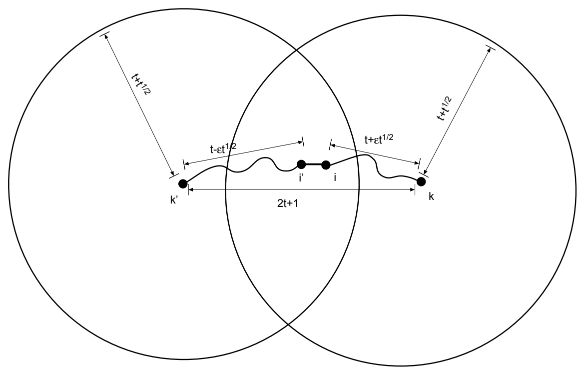

Figure 18.6: By definition, if we take two t-step random walks from two neighbors i and i', then the respective endpoints will claim the plurality assignments for u_i and u_j with probability more than $1/(2W^2)$. Because half the edges of every vertex in G have self loops, this happens even if the walks are not of length t but of length in $[t - \epsilon\sqrt{t}, t + \sqrt{t}]$ for sufficiently small ϵ.

Recall that F is the set of constraints of ψ (=edges in G) violated by the plurality assignment $\mathbf{z}$. Therefore, if $f \in F$ and $j \in \left\{t, \ldots, t + \delta\sqrt{t}\right\}$ then all the paths in $B_{j,f}$ correspond to constraints of ψ^t that are violated by the assignment $\mathbf{y}$. Therefore, we might hope that the fraction of violated constraints in ψ^t is at least the sum of $\Pr[B_{j,f}]$ for every $f \in F$ and $j \in \left\{t, \ldots, t + \delta\sqrt{t}\right\}$. If this were the case we'd be done since Claim 18.32 implies that this sum is at least $\frac{\delta\sqrt{t}\epsilon nd}{2nW^2} = \frac{\delta\sqrt{t}\epsilon}{2W^2} > \epsilon'$. However, this is inaccurate since we are overcounting paths that contain more than one such violation (i.e., paths which are in the intersection of $B_{j,f}$ and $B_{j',f'}$ for $(j,f) \neq (j',f')$). To bound the effect of this overcounting we prove the following claim:

Claim 18.33

For every $k \in \mathbb{N}$ and set F of edges with $|F| = \epsilon nd$ for $\epsilon < \frac{1}{kd}$,

$$\sum_{\substack{j,j' \in \{t..t+k\} \\ f,f' \in F \\ (j,f) \neq (j',f')}} \Pr[B_{j,f} \cap B_{j,f'}] \leq 30kd\epsilon \tag{7}$$

Proof: Only one edge can be the j^{th} edge of a path, and so for every $f \neq f'$, $\Pr[B_{j,f} \cap B_{j,f'}] = 0$. Thus the left-hand side of (7) simplifies to

$$\sum_{j \neq j' \in \{t..t+k\}} \sum_{f \neq f'} \Pr[B_{j,f} \cap B_{j',f'}] \tag{8}$$

Let A_j be the event that the j^{th} edge is in the set F. We get that (8) is equal to

$$\sum_{j \neq j' \in \{t..t+k\}} \Pr[A_j \cap A_{j'}] = 2 \sum_{j < j' \in \{t..t+k\}} \Pr[A_j \cap A_{j'}] \tag{9}$$

Let S be the set of at most $d\epsilon n$ vertices that are adjacent to an edge in F. For $j' < j$, $\Pr[A_j \cap A_{j'}]$ is bounded by the probability that a random $(j'-j)$-step path in G has both endpoints in S, or in other words that a random edge in the graph $G^{j'-j}$ has both endpoints in S. Using the fact that $\lambda(G^{j'-j}) = \lambda(G)^{j'-j} \leq 0.9^{j'-j}$, this probability is bounded by $d\epsilon(d\epsilon + 0.9^{|j-j'|})$ (see Note 18.18). Plugging this into (9) and using the formula for summation of arithmetic series, we get that:

$$2 \sum_{j < j' \in \{t,\dots,t+k\}} \Pr[A_j \cap A_{j'}] \leq$$

$$2 \sum_{j \in \{t,\dots,t+k\}} \sum_{i=1}^{t+k-j} d\epsilon(d\epsilon + 0.9^i) \leq$$

$$2k^2d^2\epsilon^2 + 2kd\epsilon \sum_{i=1}^{\infty} 0.9^i \leq 2k^2d^2\epsilon^2 + 20kd\epsilon \leq 30kd\epsilon$$

where the last inequality follows from $\epsilon < \frac{1}{kd}$. ■

Wrapping up. Claims 18.32 and 18.33 together imply that

$$\sum_{\substack{j \in \{t..t+\delta\sqrt{t}\} \\ f \in F}} \Pr[B_{j,f}] \geq \delta\sqrt{t}\epsilon\tfrac{1}{2W^2} \tag{10}$$

$$\sum_{\substack{j,j' \in \{t..t+\delta\sqrt{t}\} \\ f,f' \in F \\ (j,f) \neq (j',f')}} \Pr[B_{j,f} \cap B_{j',f'}] \leq 30\delta\sqrt{t}d\epsilon \tag{11}$$

But (10) and (11) together imply that if p is a random constraint of ψ^t then

$$\Pr[p \text{ violated by } \mathbf{y}] \geq \Pr[\bigcup_{\substack{j \in \{t..t+\delta\sqrt{t}\} \\ f \in F}} B_{j,f}] \geq \frac{\delta\sqrt{t}\epsilon}{240dW^2}$$

where the last inequality is implied by the following technical claim:

Claim 18.34

Let $A_1, \dots, A_n$ be n subsets of some set U satisfying $\sum_{i<j} |A_i \cap A_j| \leq C \sum_{i=1}^{n} |A_i|$ for some number $C \in \mathbb{N}$. Then,

$$\left| \bigcup_{i=1}^{n} A_i \right| \geq \frac{\sum_{i=1}^{n} |A_i|}{4C}$$

PROOF: We make $2C$ copies of every element $u \in U$ to obtain a set $\tilde{U}$ with $|\tilde{U}| = 2C|U|$. Now for every subset $A_i \subseteq U$, we obtain $\tilde{A}_i \subseteq \tilde{U}$ as follows: for every $u \in A_i$, we choose at random one of the $2C$ copies to put in $\tilde{A}_i$. Note that $|\tilde{A}_i| = |A_i|$. For every $i, j \in [n]$, $u \in A_i \cap A_j$, we denote by $I_{i,j,u}$ the indicator random variable that is equal to 1 if we made the same choice for the copy of u in $\tilde{A}_i$ and $\tilde{A}_j$, and equal to 0 otherwise. Since $\mathsf{E}[I_{i,j,u}] = \frac{1}{2C}$,

$$\mathsf{E}\left[|\tilde{A}_i \cap \tilde{A}_j|\right] = \sum_{u \in A_i \cap A_j} \mathsf{E}[I_{i,j,u}] = \frac{|A_i \cap A_j|}{2C}$$

and

$$\mathsf{E}\left[\sum_{i<j} |\tilde{A}_i \cap \tilde{A}_j|\right] = \frac{\sum_{i<j} |A_i \cap A_j|}{2C}$$

This means that there exists some choice of $\tilde{A}_1, \ldots, \tilde{A}_j$ such that

$$\sum_{i=1}^{n} |\tilde{A}_i| = \sum_{i=1}^{n} |A_i| \geq 2 \sum_{i<j} |\tilde{A}_i \cap \tilde{A}_j|$$

which by the inclusion-exclusion principle (see Section **??**) means that $| \cup_i \tilde{A}_i| \geq 1/2 \sum_i |\tilde{A}_i|$. But because there is a natural $2C$-to-one mapping from $\cup_i \tilde{A}_i$ to $\cup_i A_i$ we get that

$$| \cup_{i=1}^{n} A_i| \geq \frac{| \cup_{i=1}^{n} \tilde{A}_i|}{2C} \geq \frac{\sum_{i=1}^{n} |\tilde{A}_i|}{4C} = \frac{\sum_{i=1}^{n} |A_i|}{4C}$$

■

Since $\epsilon' < \frac{\delta\sqrt{t}\epsilon}{240dW^2}$, this proves the lemma. ■

18.5.2 Alphabet Reduction: Proof of Lemma 18.30

Lemma 18.30 is actually a simple consequence of Corollary 18.26, once we restate it using our "qCSP view" of **PCP** systems.

COROLLARY 18.35 (qCSP VIEW OF **PCP** OF PROXIMITY.)
There exists positive integer q_0 and an exponential-time transformation that given any circuit C of size m and and n inputs and two numbers n_1, n_2 such that $n_1 + n_2 = n$ produces an instance ψ_C of q_0CSP of size $2^{poly(m)}$ over a binary alphabet such that:

1. *The variables can be thought of as being partitioned into three sets π_1, π_2, π_3 where π_1 has 2^{n_1} variables and π_2 has 2^{n_2} variables.*

2. *If $\mathbf{u_1} \in \{0,1\}^{n_1}, \mathbf{u_2} \in \{0,1\}^{n_2}$ is such that $\mathbf{u_1} \circ \mathbf{u_2}$ is a satisfying assignment for circuit C, then there is a string π_3 of size $2^{poly(m)}$ such that $\mathsf{WH}(\mathbf{u_1}) \circ \mathsf{WH}(\mathbf{u_2}) \circ \pi_3$ satisfies ψ_C.*

3. *For every strings $\pi_1, \pi_2, \pi_3 \in \{0,1\}^*$, where π_1 and π_2 have 2^{n_1} and 2^{n_2} bits respectively, if $\pi_1 \circ \pi_2 \circ \pi_3$ satisfy at least $1/2$ the constraints of ψ_C, then π_1, π_2 are 0.99-close to $\mathsf{WH}(\mathbf{u_1})$, $\mathsf{WH}(\mathbf{u_2})$ respectively for some $\mathbf{u_1}, \mathbf{u_2}$ such that $\mathbf{u_1} \circ \mathbf{u_2}$ is a satisfying assignment for circuit C.*

Now we are ready to prove Lemma 18.30.

Proof of Lemma 18.30: Suppose the given arity 2 formula φ has n variables $u_1, u_2, \ldots, u_n$, alphabet $\{0..W{-}1\}$ and N constraints $C_1, C_2, \ldots, C_N$. Think of each variable as taking values that are bit strings in $\{0,1\}^k$, where $k = \lceil \log W \rceil$. Then if constraint C_ℓ involves variables say u_i, u_j we may think of it as a circuit applied to the bit strings representing u_i, u_j where the constraint is said to be satisfied iff this circuit outputs 1. Say m is an upperbound on the size of this circuit over all constraints. Note that m is at most $2^{2k} < W^4$. We will assume without loss of generality that all circuits have the same size.

If we apply the transformation of Corollary 18.35 to this circuit we obtain an instance of q_0CSP, say ψ_{C_l}. The strings u_i, u_j get replaced by strings of variables U_i, U_j of size $2^{2^k} < 2^{W^2}$ that take values over a binary alphabet. We also get a new set of variables that play the role analogous to π_3 in the statement of Corollary 18.35. We call these new variables Π_l.

Our reduction consists of applying the above transformation to each constraint, and taking the union of the q_0CSP instances thus obtained. However, it is important that these new q_0CSP instances *share* variables, in the following way: *for each old variable u_i, there is a string of new variables U_i of size 2^{2^k} and for each constraint C_l that contains u_i, the new q_0CSP instance ψ_{C_l} uses this string U_i.* (Note though that the Π_l variables are used only in ψ_{C_l} and never reused.) This completes the description of the new q_0CSP instance ψ (see Figure 18.7). Let us see that it works.

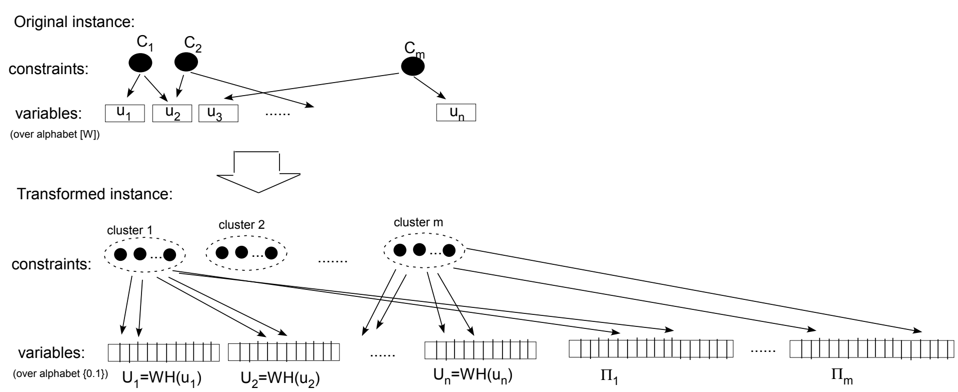

Figure 18.7: The alphabet reduction transformation maps a 2CSP instance φ over alphabet $\{0..W{-}1\}$ into a qCSP instance ψ over the binary alphabet. Each variable of φ is mapped to a block of binary variables that in the correct assignment will contain the Walsh-Hadamard encoding of this variable. Each constraint C_ℓ of φ depending on variables u_i, u_j is mapped to a cluster of constraints corresponding to all the **PCP** of proximity constraints for C_ℓ. These constraint depend on the encoding of u_i and u_j, and on additional auxiliary variables that in the correct assignment contain the **PCP** of proximity proof that these are indeed encoding of values that make the constraint C_ℓ true.

Suppose the original instance φ was satisfiable by an assignment $\mathbf{u_1}, \ldots, \mathbf{u_n}$. Then we can produce a satisfying assignment for ψ by using part 2 of Corollary 18.35, so that for each constraint C_l involving u_i, u_j, the encodings $\mathsf{WH}(\mathbf{u_i}), \mathsf{WH}(\mathbf{u_i})$ act as π_1, π_2 and then we extend these via a suitable string π_3 into a satisfying assignment for ψ_{C_l}.

On the other hand if $\mathsf{val}(\varphi) < 1 - \epsilon$ then we show that $\mathsf{val}(\psi) < 1 - \epsilon/2$. Consider any assignment $\mathbf{U_1}, \mathbf{U_2}, \ldots, \mathbf{U_n}, \mathbf{\Pi_1}, \ldots, \mathbf{\Pi_N}$ to the variables of ψ. We "decode" it to an assignment for φ as follows. For each $i = 1, 2, \ldots, n$, if the assignment to U_i is 0.99-close to a linear function, let u_i be the string encoded by this linear function, and otherwise let u_i be some arbitrary string. Since $\mathsf{val}(\varphi) < 1 - \epsilon$, this new assignment fails to satisfy at least ϵ fraction of constraints in φ. For each constraint C_l of φ that is not satisfied by this assignment, we show that at least 1/2 of the constraints in ψ_{C_l} are not satisfied by the original assignment, which leads to the conclusion that $\mathsf{val}(\psi) < 1 - \epsilon/2$. Indeed, suppose C_l involves u_i, u_j. Then $u_i \circ u_j$ is not a satisfying assignment to circuit C_l, so part 3 of Corollary 18.35 implies that regardless of the value of variables in Π_l, the assignment $\mathbf{U_i} \circ \mathbf{u_j} \circ \Pi_l$ must have failed to satisfy at least 1/2 the constraints of ψ_{C_l}. ■

18.6 The original proof of the PCP Theorem.

The original proof of the **PCP** Theorem, which resisted simplification for over a decade, used algebraic encodings and ideas that are complicated versions of our proof of Theorem 18.21. (Indeed, Theorem 18.21 is the only part of the original proof that still survives in our writeup.) Instead of the linear functions used in Welsh-Hadamard code, they use low degree multivariate polynomials. These allow procedures analogous to the linearity test and local decoding, though the proofs of correctness are a fair bit harder. The alphabet reduction is also somewhat more complicated. The crucial part of Dinur's simpler proof, the one given here, is the gap amplification lemma (Lemma 18.29) that allows to iteratively improve the soundness parameter of the **PCP** from very close to 1 to being bounded away from 1 by some positive constant. This general strategy is somewhat reminiscent of the zig-zag construction of expander graphs and Reingold's deterministic logspace algorithm for undirect connectivity described in Chapter **??**.

Chapter notes

Problems

§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 18.1 from 1/2 to s will not change the class $\mathbf{PCP}(r, q)$.

§2 Prove that for every two functions $r, q : \mathbb{N} \to \mathbb{N}$ and constant $c > 1/2$, changing the constant in the completeness condition in Definition 18.1 from 1 to c will not change the class $\mathbf{PCP}(r, q)$.

§3 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.

§4 Prove that $\mathbf{PCP}(0, \log n) = \mathbf{P}$. Prove that $\mathbf{PCP}(0, \mathrm{poly}(n)) = \mathbf{NP}$.

§5 Let L be the language of matrices A over GF(2) satisfying $\mathsf{perm}(A) = 1$ (see Chapters **??** and 8). Prove that L is in $\mathbf{PCP}(\mathrm{poly}(n), \mathrm{poly}(n))$.

§6 Show that if $\mathsf{SAT} \in \mathbf{PCP}(r(n), 1)$ for $r(n) = o(\log n)$ then $\mathbf{P} = \mathbf{NP}$. (Thus the PCP Theorem is probably optimal up to constant factors.)

§7 (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} = $ L-PCP$(\log n)$. Don't assume the PCP Theorem!

Hint: Design a verifier for 3SAT. The trivial idea would be that the proof contains a satisfying assignment and the verifier randomly picks a clause and reads the corresponding three bits in the proof to check if the clause is satisfied. This doesn't work. Why? The better idea is to require the "proof" to contain many copies of the satisfying assignment. The verifiers uses pairwise independence to run the previous test on these copies —which may or may not be the same string.

(This simple PCP Theorem is implicit in Lipton [**?**]. The suggested proof is due to van Melkebeek.)

§8 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 $J - PCP(\log n) \subseteq \mathbf{L}$.

§9 Prove that there is an **NP**-language L and $x \notin L$ such that $f(x)$ is a 3SAT formula with m constraints having an assignment satisfying more than $m - m^{0.9}$ of them, where f is the reduction from f to 3SAT obtained by the proof of the Cook-Levin theorem (Section 2.3).

Hint: show that for an appropriate language L, a slight change in the input for the Cook-Levin reduction will also cause only a slight change in the output, even though this change might cause a YES instance of the language to become a NO instance.

§10 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

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

Hint: first show that the problem can be solved exactly using dynamic programming in time poly(n, m) if all the numbers involved are in the set $[m]$. Then, show one can obtain an approximation algorithm by keeping only the $O(\log(1/\epsilon) + \log n)$ most significant bits of every number.

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

§12 Prove that QUADEQ is **NP**-complete.

Hint: show you can express satisfiability for SAT formulas using quadratic equations.

§13 Prove that if Z, U are two $n \times n$ matrices over GF(2) such that $Z \neq U$ then

$$\Pr_{\mathbf{r}, \mathbf{r}' \in_R \{0,1\}^n} [\mathbf{r} Z \mathbf{r}' \neq \mathbf{r} U \mathbf{r}'] \geq \frac{1}{4}$$

Hint: using linearity reduce this to the case that U is the all zero matrix, and then prove this using two applications of the random subsum principle.

§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.

Hint: one way to solve this is to use q-wise independent functions **??**.

§15 Let S_t be the binomial distribution over t balanced coins. That is, $\Pr[S_t = k] = \binom{t}{k} 2^{-t}$. Prove that for every $\delta < 1$, the statistical distance of S_t and $S_{t+\delta\sqrt{t}}$ is at most 10ϵ.

Hint: approximate the binomial coefficient using Stirling's formula for approximating factorials.

§16 The *long-code* for a set $\{0, \ldots, W-1\}$ is the function $\mathsf{LC} : \{0, \ldots, W-1\} \to \{0,1\}^{2^W}$ such that for every $i \in \{0..W{-}1\}$ and a function $f : \{0..W{-}1\} \to \{0,1\}$, (where we identify f with an index in $[2^w]$) the f^{th} position of $\mathsf{LC}(i)$, denoted by $\mathsf{LC}(i)_f$, is $f(i)$. We say that a function $L : \{0,1\}^{2^W} \to \{0,1\}$ is a *long-code codeword* if $L = \mathsf{LC}(i)$ for some $i \in \{0..W{-}1\}$.

(a) Prove that LC is an error-correcting code with distance half. That is, for every $i \neq j \in \{0..W-1\}$, the fractional Hamming distance of $\mathsf{LC}(i)$ and $\mathsf{LC}(j)$ is half.

(b) Prove that LC is *locally-decodable*. That is, show an algorithm that given random access to a function $L : 2^{\{0,1\}^W} \to \{0,1\}$ that is $(1-\epsilon)$-close to $\mathsf{LC}(i)$ and $f : \{0..W-1\} \to \{0,1\}$ outputs $\mathsf{LC}(i)_f$ with probability at least 0.9 while making at most 2 queries to L.

(c) Let $L = \mathsf{LC}(i)$ for some $i \in \{0..W-1\}$. Prove the for every $f : \{0..W-1\} \to \{0,1\}$, $L(f) = 1 - L(\overline{f})$, where $\overline{f}$ is the negation of f (i.e. , $\overline{f}(i) = 1 - f(i)$ for every $i \in \{0..W-1\}$).

(d) Let T be an algorithm that given random access to a function $L : 2^{\{0,1\}^W} \to \{0,1\}$, does the following:

   i. Choose f to be a random function from $\{0..W-1\} \to \{0,1\}$.
   ii. If $L(f) = 1$ then output TRUE.
   iii. Otherwise, choose $g : \{0..W-1\} \to \{0,1\}$ as follows: for every $i \in \{0..W-1\}$, if $f(i) = 0$ then set $g(i) = 0$ and otherwise set $g(i)$ to be a random value in $\{0,1\}$.
   iv. If $L(g) = 0$ then output TRUE; otherwise output FALSE.

   Prove that if L is a long-code codeword (i.e., $L = \mathsf{LC}(i)$ for some i) then T outputs TRUE with probability one.

   Prove that if L is a *linear function* that is non-zero and not a longcode codeword then T outputs TRUE with probability at most 0.9.

(e) Prove that LC is *locally testable*. That is, show an algorithm that given random access to a function $L : \{0,1\}^W \to \{0,1\}$ outputs TRUE with probability one if L is a long-code codeword and outputs FALSE with probability at least 1/2 if L is not 0.9-close to a long-code codeword, while making at most a constant number of queries to L.

   Hint: use the test T above combined with linearity testing, self correction, and a simple test to rule out the constant zero function.

(f) Using the test above, give an alternative proof for the Alphabet Reduction Lemma (Lemma 18.30).

   Hint: To transform a $2\mathsf{CSP}_W$ formula φ over n variables into a $q\mathsf{CSP}$ ψ over binary alphabet, use 2^W variables $u_j^1, \ldots, u_j^{2^W}$ for each variable u_j of φ. In the correct proof these variables will contain the longcode encoding of u_j. Then, add a set of 2^{W^2} variables $y_i^1, \ldots, y_i^{2^{W^2}}$ for each constraint φ_i of φ. In the correct proof these variables will contain the longcode encoding of the assignment for the constraint φ_i. For every constraint of φ, ψ will contain constraints for testing the longcode of both the x and y variables involved in the constraint, testing consistency between the x variables and the y variables, and testing that the y variables actually encode a satisfying assignment.

Omitted proofs

The preprocessing step transforms a $q\mathsf{CSP}$-instance φ into a "nice" $2\mathsf{CSP}$-instance ψ through the following three claims:

Claim 18.36

There is a CL- reduction mapping any $q\mathsf{CSP}$ instance φ into a $2\mathsf{CSP}_{2^q}$ instance ψ such that

$$\mathsf{val}(\varphi) \le 1-\epsilon \Rightarrow \mathsf{val}(\psi) \le 1-\epsilon/q$$

Proof: Given a $q\mathsf{CSP}$-instance φ over n variables $u_1,\ldots,u_n$ with m constraints, we construct the following $2\mathsf{CSP}_{2^q}$ formula ψ over the variables $u_1,\ldots,u_n,y_1,\ldots,y_m$. Intuitively, the y_i variables will hold the restriction of the assignment to the q variables used by the i^{th} constraint, and we will add constraints to check consistency: that is to make sure that if the i^{th} constraint depends on the variable u_j then u_j is indeed given a value consistent with y_i. Specifically, for every φ_i of φ that depends on the variables $u_1,\ldots,u_q$, we add q constraints $\{\psi_{i,j}\}_{j\in[q]}$ where $\psi_{i,j}(y_i,u_j)$ is true iff y_i encodes an assignment to $u_1,\ldots,u_q$ satisfying φ_i and u_j is in $\{0,1\}$ and agrees with the assignment y_i. Note that the number of constraints in ψ is qm.

Clearly, if φ is satisfiable then so is ψ. Suppose that $\mathsf{val}(\varphi) \le 1-\epsilon$ and let $u_1,\ldots,u_k,y_1,\ldots,y_m$ be any assignment to the variables of ψ. There exists a set $S \subseteq [m]$ of size at least ϵm such that the constraint φ_i is violated by the assignment $u_1,\ldots,u_k$. For any $i \in S$ there must be at least one $j \in [q]$ such that the constraint $\psi_{i,j}$ is violated. ■

Claim 18.37

There is an absolute constant d and a CL- reduction mapping any $2\mathsf{CSP}_W$ instance φ into a $2\mathsf{CSP}_W$ instance ψ such that

$$\mathsf{val}(\varphi) \le 1-\epsilon \Rightarrow \mathsf{val}(\psi) \le 1-\epsilon/(100Wd).$$

and the constraint graph of ψ is d-regular. That is, every variable in ψ appears in exactly d constraints.

Proof: Let φ be a $2\mathsf{CSP}_W$ instance, and let $\{G_n\}_{n\in\mathbb{N}}$ be an explicit family of d-regular expanders. Our goal is to ensure that each variable appears in φ at most $d+1$ times (if a variable appears less than that, we can always add artificial constraints that touch only this variable). Suppose that u_i is a variable that appears in k constraints for some $n > 1$. We will change u_i into k variables $y_i^1,\ldots,y_i^k$, and use a different variable of the form y_i^j in the place of u_i in each constraint u_i originally appeared in. We will also add a constraint requiring that y_i^j is equal to $y_i^{j'}$ for every edge (j,j') in the graph G_k. We do this process for every variable in the original instance, until each variable appears in at most d equality constraints and one original constraint. We call the resulting $2\mathsf{CSP}$-instance ψ. Note that if φ has m constraints then ψ will have at most $m+dm$ constraints.

Clearly, if φ is satisfiable then so is ψ. Suppose that $\mathsf{val}(\varphi) \le 1-\epsilon$ and let $\mathbf{y}$ be any assignment to the variables of ψ. We need to show that $\mathbf{y}$ violates at least $\frac{\epsilon m}{100W}$ of the constraints of ψ. Recall that for each variable u_i that appears k times in φ, the assignment $\mathbf{y}$ has k variables $y_i^1,\ldots,y_i^k$. We compute an assignment $\mathbf{u}$ to φ's variables as follows: u_i is assigned the plurality value of

$y_i^1, \ldots, y_i^k$. We define t_i to be the number of y_i^j's that *disagree* with this plurality value. Note that $0 \leq t_i \leq k(1 - 1/W)$ (where W is the alphabet size). If $\sum_{i=1}^n t_i \geq \frac{\epsilon}{4}m$ then we are done. Indeed, by (3) (see Note 18.18), in this case we will have at least $\sum_{i=1}^n \frac{t_i}{10W} \geq \frac{\epsilon}{40W}m$ equality constraints that are violated.

Suppose now that $\sum_{i=1}^n t_i < \frac{\epsilon}{4}m$. Since $\mathsf{val}(\varphi) \leq 1-\epsilon$, there is a set S of at least ϵm constraints violated in φ by the plurality assignment $\mathbf{u}$. All of these constraints are also present in ψ and since we assume $\sum_{i=1}^n t_i < \frac{\epsilon}{4}m$, at most half of them are given a different value by the assignment $\mathbf{y}$ than the value given by $\mathbf{u}$. Thus the assignment $\mathbf{y}$ violates at least $\frac{\epsilon}{2}m$ constraints in ψ. ■

Claim 18.38

There is an absolute constant d and a CL-reduction mapping any $\mathsf{2CSP}_W$ instance φ with d'-regular constraint graph for $d \geq d'$ into a $\mathsf{2CSP}_W$ instance ψ such that

$$\mathsf{val}(\varphi) \leq 1 - \epsilon \Rightarrow \mathsf{val}(\psi) \leq 1 - \epsilon/(10d)$$

and the constraint graph of ψ is a $4d$-regular expander, with half the edges coming out of each vertex being self-loops.

Proof: There is a constant d and an explicit family $\{G_n\}_{n\in\mathbb{N}}$ of graphs such that for every n, G_n is a d-regular n-vertex 0.1-expander graph (See Note 18.18).

Let φ be a $\mathsf{2CSP}$-instance as in the claim's statement. By adding self loops, we can assume that the constraint graph has degree d (this can at worst decrease the gap by factor of d). We now add "null" constraints (constraints that always accept) for every edge in the graph G_n. In addition, we add $2d$ null constraints forming self-loops for each vertex. We denote by ψ the resulting instance. Adding these null constraints reduces the fraction of violated constraints by a factor at most four. Moreover, because any regular graph H satisfies $\lambda(H) \leq 1$ and because of λ's subadditivity (see Exercise 11, Chapter **??**), $\lambda(\psi) \leq \frac{3}{4} + \frac{1}{4}\lambda(G_n) \leq 0.9$ where by $\lambda(\psi)$ we denote the parameter λ of ψ's constraint graph. ■