Probabilistic Checking of Proofs and Hardness of Approximation Problems

Sanjeev Arora

CS-TR-476-94

(Revised version of a dissertation submitted at
CS Division, UC Berkeley, in August 1994)

Probabilistic Checking of Proofs
and
Hardness of Approximation Problems

by
Sanjeev Arora

The dissertation committee consisted of:

Professor Umesh V. Vazirani (Chair)
Professor Richard M. Karp
Professor John W. Addison

The author's current address:

Department of Computer Science
35 Olden St.
Princeton University
Princeton, NJ 08544
email: arora@cs.princeton.edu

To my parents,
and
members of my family
(including the ones who just joined)

Contents

List of Figures

Acknowledgements

I must acknowledge my great debt to my advisor Umesh Vazirani for the nurturing and advising, the sake, and the dinners he has provided me over the last four years. My only regret is that I never took squash lessons from him even though I kept planning to.

The theory group at Berkeley was a wonderful enviroment in which to be a graduate student. Thanks to Dick Karp, Manuel Blum, Mike Luby, Raimund Seidel, Gene Lawler, and Abhiram Ranade for creating the environment. Dick's phenomenally clear mind proved a good resource on many occasions. Manuel and Mike got me thinking about many of the problems upon which this dissertation is based. Some of Mike's unpublished observations were central to guiding my dissertation research in the right direction.

I would like to thank my colleagues with whom I did this research. Special thanks go to Madhu Sudan and Muli Safra for their patience while collaborating with me in my early grad-school days. I learnt a lot from them. I also thank Carsten Lund, Jacques Stern, Laci Babai, Mario Szegedy, Rajeev Motwani, and Z Sweedyk for collaborating with me.

Thanks to John Addison for serving on my dissertation committee and giving me lots of comments in his usual precise style.

Thanks to all my fellow-students in the department, and the post-docs at ICSI, for talks, TGIF's, happy hours, and bull-sessions.

Thanks also to all the people at MIT who got me interested in theoretical computer science and softball during my undergraduate years. I am especially grateful to Bruce Maggs, Tom Leighton, Mike Sipser, Mauricio Karchmer, Richard Stanley, Charles Leiserson, and David Shmoys.

There are many others who over the years have helped me develop as a person and a researcher. I will not attempt to list their names, for fear that any such list will be incomplete, but I express my thanks to them all.

My deepest gratitude is to my family for instilling a love of knowledge in me. I hope my father is not too dismayed that I am contributing two more professors to the family.

Finally, I would like to thank Silvia. She was clearly my best discovery at Berkeley.

Chapter 1

Introduction

The study of the difficulty (or hardness) of computational problems has two parts. The *theory of algorithms* is concerned with the design of efficient algorithms; in other words, with proving upper bounds on the amount of computational resources required to solve a specific problem. *Complexity theory* is concerned with proving the corresponding lower bounds. Our work is part of the second endeavor; more specifically, the endeavor to prove problems computationally difficult, or *hard*.

Despite some notable successes, lower bound research is still in a stage of infancy; progress on its open problems has been slow. Central among these open problems is the question whether $P \neq NP$. In other words, is there a problem that can be solved in polynomial time on a nondeterministic Turing Machine, but cannot be solved in polynomial time deterministically? The conjecture $P \neq NP$ is widely believed, but currently our chances of proving it appear slim.

If we assume that $P \neq NP$, however, then another interesting question arises: given any specific optimization problem of interest, is it in P or not? Complexity theory has had remarkable success in answering such questions. The theory of *NP-completeness* due to Cook, Levin, and Karp allows us to prove that explicit problems are not in P, assuming $P \neq NP$. The main idea is to prove the given problem *NP-hard*, that is, to give a polynomial-time reduction from instances of any NP problem to instances of the given problem. If an NP-hard problem were to have a polynomial-time algorithm, so would every NP problem, which would contradict the assumption $P \neq NP$. Hence if $P \neq NP$ then an NP-hard problem has no polynomial-time algorithm. (To put it differently, an NP-hard problem is no easier than any other problem in NP.)

The success of the theory of NP-completeness lay in the unity it brought to the study of computational complexity: a wide array of optimization problems arising in practice (and which had hitherto defied all efforts of algorithm designers to find efficient algorithms) were proved NP-hard in one swoop, using essentially the same kind of reductions. (For a list of NP-hard problems c. 1979, see the survey by Garey and Johnson [GJ79].)

But one major group of problems seemed not to fit in the framework of NP-completeness: approximation problems. *Approximating* an NP-hard optimization problem *within a factor* c means to compute solutions whose "cost" is within a multiplicative factor c of the cost of the optimal solution. Such solutions would suffice in practice, if c were close enough to 1. For some NP-hard problems we know how to compute such solutions in polynomial time; for most it seemed that even approximation was hard – at least, a substantial body of research failed to yield any efficient approximation algorithms. However, it was not clear how to prove the hardness of approximation using the Cook-Karp-Levin technique. (We elaborate on this point in Chapter 2.) A recent result by Feige et al ([FGL$^+$91]) provided a breakthrough, by using algebraic techniques to show that approximating the clique problem is hard. These algebraic techniques are derived from recent work on interactive proofs and program checking (see Section 2.5).

1.1. This dissertation

We use techniques similar to those in the above-mentioned paper ([FGL$^+$91]) to prove the hardness of many other approximation problems. We also prove a new probabilistic characterization of the class NP. The results in this dissertation are from three papers ([AS92, ALM$^+$92, ABSS93]) and some previously unpublished observations.

1.1.1. Hardness of Approximation

We exhibit many NP-hard optimization problems for which approximation (for a range of values of the factor c) is NP-hard. In other words, approximating the problem is no easier than solving it exactly (at least as far as polynomial-time solvability is concerned). Some of the important problems to which our result applies are the following.

Clique and Independent Set. We show that approximating these problems within any constant factor is NP-hard ([AS92]). Further, in [ALM$^+$92] we show that some positive constant ϵ exists such that approximating these problems within a factor of n^ϵ (n = number of vertices in the graph) is NP-hard. A weaker hardness result was known earlier, namely, that if these problems can be approximated within any constant factor in polynomial time, then all NP problems can be solved deterministically in time $n^{O(\log\log n)}$ ([FGL$^+$91]).

MAX-SNP-hard problems. The class MAX-SNP of optimization problems was identified by Papadimitriou and Yannakakis ([PY91]), and the class of problems *hard* for this class include vertex-cover, metric TSP, shortest superstring, and others. We show that for every MAX-SNP-hard problem there is some fixed constant $c > 1$ such that approximating the problem within a factor c is NP-hard ([ALM$^+$92]).

Optimization problems on lattices, codes and linear systems. We show the hardness of approximating many problems including the well-known Nearest Lattice Vector and the Nearest Codeword problems [ABSS93]. A hardness result is also obtained for a version of the Shortest Lattice Vector problem, namely the version using the ℓ_∞ norm. We note that tightening the ℓ_∞ result would prove the hardness of exact optimization in the ℓ_2 norm, a longstanding open problem.

A self-contained description of the above hardness results appears in Chapter 6. That chapter also attempts to unify all known hardness results. More specifically, it introduces two canonical problems and indicates how reductions from them can be used to prove all known hardness results. For some problems however, this approach yields inapproximability results not as strong as those provable otherwise. Chapter 7 discusses a possible approach to remedy this difficulty.

1.1.2. A New Characterization of NP

Our results on hardness of approximation rely on a new type of reduction whose main feature is that it acts globally, unlike classical reductions, which perform local transformations. Another way to view this new reduction is as a new definition of NP. According to this definition, the class NP contains exactly those languages for which membership proofs can be checked by a probabilistic verifier that uses $O(\log n)$ random bits and examines $O(1)$ bits in the proof ([AS92, ALM$^+$92]). (Please see Chapter 2 for a more careful statement.) The equivalence between the new and the old definitions of NP is the subject of the so-called the PCP Theorem, whose proof uses algebraic techniques partially derived from previous work. An outline of the proof appears in Chapter 3, and details appear in Chapters 4 and 5.

At the end of each chapter, a brief section gives pointers to other literature, and a historical account of the the development of the ideas of that chapter.

To the best of our knowledge, this dissertation represents the first self-contained exposition of the entire proof of the PCP Theorem, incorporating all necessary lemmas from the papers ([AS92, ALM$^+$92]), and other previous work. For other (almost complete) expositions we refer the reader to [Sud92, ALM$^+$92]). However, the exposition in [Sud92] takes a different viewpoint: Its main results concern program checking, and the PCP theorem is derived as a corollary to those results.

We feel that in the long run the algebraic techniques used in proving the PCP theorem will find many other applications. To some extent, this has already happened, and in Chapter 8 we include a brief survey of some of the recently-discovered applications. (Many applications are due to other researchers.)

Finally, Chapter 9 contains a list of open questions about both hardness of approximation, and the algebraic techniques used in earlier chapters. One important open question is whether proof of the PCP theorem can be simplified. Chapter 9 discusses this question as well.

1.1.3. Knowledge assumed of the Reader

This dissertation has been written as a survey for the nonspecialist. We assume only familiarity with Turing Machines (and standard conventions about them), asymptotic notation and polynomial time, and NP-completeness. For an introduction to all these see [GJ79]. A list of assumed algebraic facts, with brief proofs, appears in Appendix A. However, most readers should find that they can understand most of the dissertation on the basis of just the following mantra.

A non-zero univariate polynomial of degree d has at most d roots in a field.

Chapter 2

Old vs. New Views of NP

Let Σ be a finite set (called the *alphabet*). A *language* L is a set of finite-sized strings over Σ, i.e., $L \subseteq \Sigma^*$. By a straightforward equivalent of the classical definition, a language L is in NP iff there is a polynomial time deterministic Turing Machine M and a positive number c such that for any x in Σ^*:

$$x \in L \text{ iff } \exists y \in \Sigma^*, |y| = |x|^c, \text{ s.t. } M \text{ accepts } (x, y). \tag{2.1}$$

String y is variously called "witness", "nondeterministic guess," or "membership proof." We prefer the term "membership proof." Machine M is called the *verifier*.

As is standard, we assume $\Sigma = \{0, 1\}$.

Example 2.1: A 3CNF formula in boolean variables $s_1, \ldots, s_n$ is of the form

$$\bigwedge_{i=1}^{m} (w_{i_1} \vee w_{i_2} \vee w_{i_3}),$$

where each w_{i_j} is a literal, i.e., either s_k or $\neg s_k$ for some k. The subformula $(w_{i_1} \vee w_{i_2} \vee w_{i_3})$ is called a *clause*. The formula is *satisfiable* if there is an assignment to the s_i's which makes all clauses true.

Let 3SAT be the set of satisfiable 3CNF formulae. By encoding formulae with 0s and 1s in some canonical fashion, we can consider 3SAT as a language. It is in NP, since a satisfying assignment constitutes a membership proof that can be checked in polynomial time.

2.1. The Old View: Cook-Levin Theorem

Cook ([Coo71]), and independently, Levin ([Lev73]) showed that 3SAT is NP-complete. More specifically, given any NP-language L and input x, they gave a polynomial time

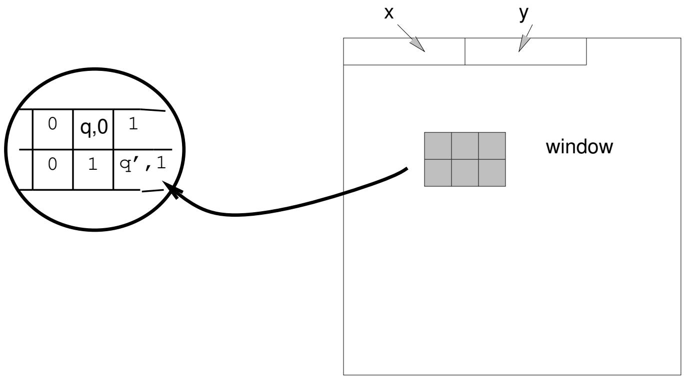

Figure 2.1: Tableau and "window". The window shows the finite control initially in state q and reading a 0. The control overwrites the 0 with a 1, moves one cell to right, and changes state to q'.

construction of a 3CNF formula $\varphi_{x,L}$ that is satisfiable iff $x \in L$. The Cook-Levin result underlies classical work on NP-completeness, since most other problems are proven NP-complete by doing reductions from 3SAT. We briefly recall the textbook version of their construction (for further details, see [GJ79]).

How can a generic reduction be given from all NP languages to 3SAT? After all, the notion of membership proof differs widely for different NP languages. Cook and Levin noticed that a single notion (or format) suffices for all languages: The proof can be the tableau of an accepting computation of the polynomial-time verifier. Further, in this format, the proof is correct iff it satisfies some local constraints.

A *tableau* is a 2-dimensional transcript of a computation. If the computation ran for T steps, the tableau is a 2-dimensional table that has T lines of T entries each, where jth entry in line i contains the following information : (i) the contents of the jth cell of the tape at time i; and (ii) whether or not the finite control was in that cell or not, and if so, what state it was in. (see Figure 2.3 in [GJ79]).

Let L be an NP-language, and M be the verifier for L. A tableau is *valid for M* if each step of the computation followed correctly from the previous step, and the verifier is in an accept state in the last line.

A look at the definition in (2.1) shows that an input x is in L iff

$$\exists \text{ valid tableau for } M \text{ with first line } (x, y) \text{, for a string } y \text{ of a suitable size.} \qquad (2.2)$$

Given a tableau, here is how to check that it satisfies the conditions in (2.2). (a) Check bit-by-bit that the first line contains x. (b) Check that M is in an accept state in the last line. And finally, (c) check that the computation was done correctly at each step.

But to check (c) we only need to check that the finite control of M operated correctly at each step, and that every tape cell not in the immediate neighborhood of the finite control does not change its contents in that step.

Thus whether or not the tableau passes the checks (a), (b), and (c) depends upon whether or not its entries satisfy some local constraints. In fact, it is easy to see that the computation is done correctly overall iff all 2×3 neighborhoods in the tableau look "correct." This 2×3 neighborhood is sometimes called a "window." (See Figure 2.1.)

Here is how Cook and Levin constructed a 3CNF formula that is satisfiable iff there is a tableau that passes the checks (a), (b) and (c). For $i, j \leq T$, represent the contents of the the jth entry of the ith line by $O(1)$ boolean variables. For each window, express its correctness by a boolean function of the variables corresponding to the cells in that window. Rewrite this function using clauses of size 3.

The overall formula is the $\wedge$ of the formulae expressing checks (a), (b), and (c). For instance the formula expressing (c) is

$$\bigwedge_{i,j \leq T} (\text{Formula expressing correctness of window around } j\text{th cell of } i\text{th line}).$$

Example 2.2: *(We give some details of the Cook-Levin construction; see [GJ79] for further details.)* Assume the machine's alphabet is $\{0, 1\}$. The corresponding formula has for each $i, j \leq T$ the variables z_{ij}, y_{ij} and for each state q in the finite control, a variable s_{ijq}. The interpretation to $z_{ij} = y_{ij} = b$ for $b \in \{0, 1\}$ is: " At time i the the jth cell contains bit b." And $z_{ij} \neq y_{ij}$ means the cell has a blank symbol. The interpretation to s_{ijq} being set to true is: "At time i the jth cell contains the Turing Machine head, and the finite control is in state q." Then here's how to express that if the finite control is in cell j at time i, then it cannot be in multiple states.

$$\wedge_{q \neq q'} (\neg s_{ijq} \vee \neg s_{ijq'}).$$

2.2. Optimization and Approximation

With most well-known NP languages we can associate a natural optimization problem. The problem associated with 3SAT is MAX-3SAT. For a 3CNF formula ϕ, and assignment A, let *val*(A/ϕ) denote the number of clauses in ϕ satisfied by A.

Definition 2.1 (MAX-3SAT): This is the following problem.
Input: 3CNF formula ϕ.
Output: OPT(ϕ), which is $\max\{val(A/\phi) : A \text{ is an assignment}\}$.

Clearly, MAX-3SAT is NP-hard. As mentioned in the introduction, a natural way to deal with NP-hardness is to compute approximate solutions.

Definition 2.2: For a rational number $c > 1$, an algorithm is said to *compute c-approximations* to MAX-3SAT if given any input ϕ its output is an assignment B such that

$$\frac{\mathrm{OPT}(\phi)}{c} \leq val(B/\phi) \leq \mathrm{OPT}(\phi).$$

This dissertation addresses the following question: For what values of c can c-approximations to MAX-3SAT be computed in polynomial time? For $c = 4/3$ this was known to be possible ([Yan92], see also [GW94]). (The algorithm for $c = 2$ is actually quite straightforward.) Whether or not the same was true for every fixed constant $c > 1$ was not known.

The Cook-Levin reduction does not rule out the existence of polynomial-time algorithms that compute c-approximations for every fixed $c > 1$. Recall that the 3CNF formulae it produces always represent tableaus. For such formulae we show how to satisfy a fraction $(1 - 1/T)$ of the clauses in polynomial time, where T is the number of lines in the tableau. Construct in polynomial time an invalid tableau that starts off by representing a valid computation on (x, y) for some string y, but then "switches" the computation to some trivial accepting computation. In such a tableau all the "windows" look correct except those in the line where the switch was made. Now interpret the tableau as an assignment to the corresponding 3CNF formula. The assignment satisfies all the clauses except the ones corresponding to a single line; in other words, all but $1/T$ fraction of the clauses. Since $1/T$ is smaller than any constant, we conclude that in the Cook-Levin instances of MAX-3SAT, optimization is hard, but computing c-approximations for any fixed $c > 1$ is easy.

Furthermore, the known reductions from 3SAT to other optimization problems transform 3SAT instances in a local fashion, namely, by using "gadgets" to represent clauses and variables. When performed on the Cook-Levin instances of 3SAT, such local transformations yield instances of the other problem in which optimization is hard but c-approximation is easy for every $c > 1$.

Thus it becomes clear that a new type of NP-completeness reduction is called for to prove the hardness of approximations.

2.3. A New View of NP

In this section we state a new probabilistic definition of NP. It is based upon a new complexity class, PCP, whose name abbreviates **P**robabilistically **C**heckable **P**roofs.

Let a *verifier* be a polynomial-time probabilistic Turing Machine containing an input tape, a work tape, a source of random bits, and a read-only tape called the *proof string* and denoted as Π (see Figure 2.2). The machine has *random access* to Π: the work-tape contains a special *addressing portion* on which M can write the address of a location in Π, and then read just the bit in that location. The operation of reading a bit in Π is called a *query*.

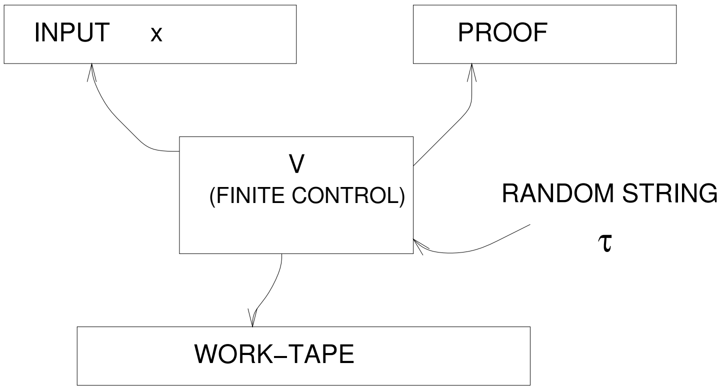

Figure 2.2: Verifier in the definition of PCP.

The source of random bits generates at most one bit per step of the machine's computation. Since the machine uses only a finite sequence of those bits, we can view the sequence as an additional input to the machine, called the *random string.*

Definition 2.3: A verifier is $(r(n), q(n))$-*restricted* if on each input of size n it uses at most $O(r(n))$ random bits for its computation, and queries at most $O(q(n))$ bits of the proof.

In other words, an $(r(n), q(n))$-restricted verifier has two associated integers c, k. The random string has length $cr(n)$. The verifier operates as follows on an input of size n. It reads the random string τ, computes a sequence of $k\,q(n)$ locations $i_1(\tau), \ldots, i_{k\,q(n)}(\tau)$, and queries those locations in Π. Depending upon what these bits were, it accepts or rejects[1].

Define $M^{\Pi}(x, \tau)$ to be 1 if M accepts input x, with access to the proof Π, using a string of random bits τ, and 0 otherwise.

Definition 2.4: A verifier M can *probabilistically check membership proofs* for language L if

- For every input x in L, there is a proof Π_x that causes M to accept for every random string (i.e. with probability 1).
- For any input x not in L, every proof Π is rejected with probability at least 1/2.

$$\Pr_{\tau}[M^{\Pi}(x, \tau) = 1] < \frac{1}{2}.$$

[1]Note that we are restricting the verifier to query the proof *non-adaptively*: the locations it queries in Π depend only upon its random string. In contrast, the original definition of PCP ([AS92]) allowed the verifier base its next query on the bits it had already read from Π. We include nonadaptiveness as part of the definition because many applications require it.

Note: The choice of probability 1/2 in the second part is arbitrary. By repeating the verifier's program $O(1)$ times, (and rejecting if the verifier rejects even once) the probability of rejection 1/2 in the second part can be reduced to any arbitrary positive constant. Thus we could've used any constant less than 1 instead of 1/2.

Definition 2.5: A language L is in PCP$(r(n), q(n))$ if there is an $(r(n), q(n))$-restricted verifier M that can check membership proof for L.

Note that NP = PCP$(0, \text{poly}(n))$, since PCP$(0, \text{poly}(n))$ is the set of languages for which membership proofs can be checked in deterministic polynomial-time, which is exactly NP, according to Definition (2.1). In Chapter 3 we will prove the following result.[2]

Theorem 2.1 (PCP Theorem): *$NP = PCP(\log n, 1)$.*

Next, we prove the easier half of the PCP Theorem: PCP$(\log n, 1) \subseteq$ NP. Observe that when the input has size n, a $(\log n, 1)$-restricted verifier has $2^{O(\log n)} = n^{O(1)}$ choices for its random string. Further, once we fix the random string, the verifier's decision is based upon $O(1)$ bits in the proof. More formally, we can state the following lemma.

Lemma 2.2: *Let language L be in PCP$(\log n, 1)$. Then there are integers $c, d, k \geq 1$ such that for every input x there are n^c boolean functions $f_1, f_2, \ldots$ in n^d boolean variables (where n is the size of x) with the following properties.*

1. *Each f_i is a function of only k variables, and its truth-table can be computed in polynomial time given (x, i).*
2. *If input x is in L, there is an assignment to the boolean variables that makes every f_i evaluate to true.*
3. *If $x \notin L$ then no assignment to the boolean variables makes more than 1/2 the f_i's true.*

Proof: Let the verifier for L use $c \log n$ random bits. Note that it has at most $2^{c \log n} = n^c$ different possible runs, and in each run it reads only $O(1)$ bits in the proof-string. Hence w.l.o.g. we can assume that the number of bits in any provided proof-string is at most $O(n^c)$. For concreteness, assume this number is n^d for some integer d.

For any boolean-valued variables $y_1, \ldots, y_{n^d}$, the set of possible assignments to $y_1, \ldots, y_{n^d}$ is in one-to-one correspondence with the set of possible proof-strings. We assume w.l.o.g. that the proof-string is an assignment to the variables $y_1, y_2, \ldots, y_{n^d}$.

[2] John Addison has pointed out an (unintended) pun in this result. In descriptive set theory, PC$\mathcal{C}$ for any class $\mathcal{C}$ would be the "projections of sets that are complements of sets of $\mathcal{C}$." For $\mathcal{C}$ = the complexity class P, this would refer to NP, since complement of P is P and projection of P is NP.

Fixing the verifier's random string to $r \in \{0,1\}^{c \log n}$, fixes the sequence of locations that it will examine in the proof. Let this sequence have size k ($= O(1)$). Let the sequence of locations be $i_1(r), \ldots, i_k(r)$. The verifier's decision depends only upon the assignments to $y_{i_1(r)}, \ldots, y_{i_k(r)}$. Define a boolean function on k bits, f_r, as $f_r(b_1, \ldots, b_k) =$ true iff the verifier accepts when the assignment to the sequence of variables $y_{i_1(r)}, \ldots, y_{i_k(r)}$ is $b_1, \ldots, b_k$. Since the verifier runs in polynomial time, we can compute the truth table of f_r in polynomial time by going through all possible 2^k values of $b_1, \ldots, b_k$, and computing the verifier's decision on each sequence.

Consider the set of n^c functions $\left\{f_r : r \in \{0,1\}^{c \log n}\right\}$ defined in this fashion. By definition of PCP($\log n, 1$), when the input is in the language, there is an assignment to the $y_1, y_2, \ldots,$ that makes all functions in this set evaluate to true. Otherwise no assignment makes more than 1/2 of them evaluate to true. $\Box$

Corollary 2.3: *PCP*($\log n, 1$) $\subseteq$ *NP*.

Proof: Let $L \in$ PCP($\log n, 1$), and x be an input of size n. Construct in poly(n) time the functions of Lemma 2.2. Clearly, $x \in L$ iff there exists an assignment to the n^d variables that makes all the f_i's evaluate to TRUE. Such an assignment constitutes a "membership proof" that $x \in L$, and can be checked in poly(n) time. $\Box$

2.4. The PCP Theorem: Connection to Approximation

This section shows that the characterization of NP as PCP($\log n, 1$) allows us to define a new format for membership proofs for NP languages. The format is more robust (in a sense explained below) than the tableau format of Cook and Levin, and immediately suggests a new way to reduce NP to MAX-3SAT. This new reduction shows that approximating MAX-3SAT is NP-hard.

Let L be any NP language. Since NP $\subseteq$ PCP($\log n, 1$), Lemma 2.2 holds for it. Let x be an input. Use the set of functions given by Lemma 2.2 to define a new format for membership proofs for L: the proof is a boolean assignment (i.e., a sequence of bits) that makes all of $f_1, f_2, \ldots,$ evaluate to true.

Then follow the Cook-Levin construction closely. Think of each f_i as representing a "correctness condition" for a set of k bit-positions in the proof; thus this set of k locations plays a role analogous to that of a "window" in the tableau. The statement of Lemma 2.2 implies that if $x \notin L$ then *half the windows are incorrect*. To contrast the new format with the tableau format, recall that when $x \notin L$ there exist tableaus in which almost all the windows look correct. In this sense the new format is more robust.

Formally, the reduction consists in writing a 3SAT formula that represents all the f_i's. The gap (of a factor 2) between the fraction of f_i's that can be satisfied in the two cases translates into a gap between the fraction of clauses that can be satisfied in the two case.

The proof of the following corollary formalizes the above description.

Corollary 2.4: *There is a constant* $\epsilon > 0$ *such that computing* $(1 + \epsilon)$*-approximations to MAX-3SAT is NP-hard.*

Proof: Let L be an NP-complete language and x be an input of size n. Assuming NP $\subseteq$ PCP$(\log n, 1)$, Lemma 2.2 applies to L. We use notation from that lemma. Let $y_1, \ldots, y_{n^d}$ be the set of boolean variables and $\{f_i : 1 \leq i \leq n^c\}$ the collection of functions corresponding to x.

Consider a function f_i from this collection. Let it be a function of variables $y_{i_1}, \ldots, y_{i_k}$. Then f_i can be expressed as a conjunction of 2^k clauses in these variables, each of size at most k. Let $C_{i,1}, \ldots, C_{i,2^k}$ denote these clauses. (From now on we use the terms k-clause and 3-clause to talk about clauses of size k and 3 respectively.)

Then the k-CNF formula

$$\bigwedge_{i=1}^{n^c} \bigwedge_{j=1}^{2^k} C_{i,j} \tag{2.3}$$

is satisfiable iff $x \in L$. Also, if $x \notin L$, then every assignment fails to satisfy half the f_i's, each of which yields an unsatisfied k-clause. So if $x \notin L$ the fraction of unsatisfied clauses is at least $\frac{1}{2} \times \frac{1}{2^k}$, which is some fixed constant.

To obtain a 3CNF formula rewrite every k-clause as a conjunction of clauses of size 3, as follows. For a k-clause $l_1 \vee l_2 \vee \cdots \vee l_k$ (the l_i's are literals), write the formula

$$(l_1 \vee l_2 \vee z_1) \wedge (l_{k-1} \vee l_k \vee \neg z_{k-2}) \wedge \bigwedge_{t=1}^{k-4} (\neg z_t \vee l_{t+2} \vee z_{t+1}) \tag{2.4}$$

where $z_1, \ldots, z_{k-2}$ are new variables which are not to be used again for any other k-clause. Clearly, a fixed assignment to $l_1, \ldots, l_k$ satisfies the original k-clause iff there is a further assignment to $z_1, \ldots, z_{k-2}$ that satisfies the formula in (2.4).

Thus the formula of (2.3) has been rewritten as a 3CNF formula that is satisfiable iff $x \in L$. Further, if $x \notin L$, every unsatisfied k-clause in (2.3) yields an unsatisfied 3-clause in the new formula, so the fraction of unsatisfied 3-clauses is at least $\frac{1}{k-2} \times \frac{1}{2^{k+1}}$.

Hence the lemma has been proved for the value of ϵ given by

$$\frac{1}{1+\epsilon} = 1 - \frac{1}{(k-2)2^{k+1}}.$$

$\square$

As shown in Chapter 6, Corollary 2.4 implies similar hardness results for a host of other problems.

Where the gap came from. The gap of a factor of ϵ in the above reduction came from two sources: the gap 1 versus 1/2 in the fraction of satisfiable f_i's in Lemma 2.2, and the fact that each f_i involves $O(1)$ variables. But recall that each f_i just represents a possible run of the $(\log n, 1)$-restricted verifier for the language. Thus the description of each f_i depends upon the construction of the verifier. Unfortunately, the only known construction of this verifier is quite involved. It consists in defining a complicated algebraic object, which exists iff the input is in the language. The verifier expects a membership proof to contain a representation of this object. In each of its runs the verifier examines a different part of this provided object. Thus the function f_i representing that run is a correctness condition for that part of the object.

For details on the algebraic object, we refer the reader to the next chapter. A detail worth mentioning is that each part of the object – and thus, the definition of each f_i – depends upon every input bit. This imparts our reduction a global structure. In contrast, classical NP-completeness reductions usually perform local transformations of the input.

2.5. History and Background

Approximability. The question of approximability started receiving attention soon after NP-completeness was discovered [Joh74, SG76]. (See [GJ79] for a discussion). Much of the work attempted to discover a classification framework for optimization (and aproximation) problems analogous to the framework of NP-completeness for decision problems. (See [ADP77, ADP80, AMSP80] for some of these attempts.) The most successful attempt was due to Papadimitriou and Yannakakis, who based their classification around a complexity class they called MAX-SNP (see Chapter 6). They proved that MAX-3SAT is complete for MAX-SNP, in other words, any unapproximability result for MAX-3SAT transfers automatically to a host of other problems. The desire to prove such an unapproximability result motivated the discovery of the PCP theorem.

Roots of PCP. The roots of the definition of PCP (specifically, the fact that the verifier is randomized) go back to the definition of Interactive Proofs (Goldwasser, Micali, and Rackoff [GMR89]) and Arthur-Merlin games (Babai[Bab85], see also [BM88]). Two complexity classes arise from their definitions: IP and AM respectively. Both feature a polynomial time verifier interacting with an all-powerful adversary, called the prover, who has to convince the verifier that the given input is in the language. (The difference between the two classes is that in the definition of IP, the prover cannot read the verifier's random string.) Early results about these classes fleshed out their properties, including the surprising fact that they are the same class ([GS86]), see also [GMS87]).

The next step involved the invention of multi-prover interactive proofs, and the associated complexity class MIP by Ben-Or, Goldwasser, Killian, and Wigderson ([BGKW88]). Here the single all-powerful prover in the IP scenario is replaced by many all-powerful provers who cannot communicate with one another during the protocol. Again, the motivation was cryptography, although soon Fortnow, Rompel and Sipser [FRS88] analyzed the new model from a complexity-theoretic viewpoint. They proved that the class MIP is exactly the class of languages for which membership proofs can be checked by a probabilistic polynomial time verifier that has random access to the proof. Since the probabilistic verifier can access (over all choices of its random seed) a proof of exponential size, it follows that MIP $\subseteq$ NEXPTIME. Recall that NEXPTIME is the exponential analogue of NP. It contains the set of languages that can be accepted by a nondeterministic Turing Machine that runs in exponential time. Since it seemed "clear" that MIP was quite smaller than NEXPTIME, the statement MIP $\subseteq$ NEXPTIME was considered unsatisfactorily weak. (A similar situation prevailed for IP, where the analogous statement, IP $\subseteq$ PSPACE –proved implicitly by Papadimitriou in [Pap83]– was considered quite weak.)

The next development in this area came as a big surprise. Techniques from program-checking (due to Blum and Kannan [BK89], Lipton [Lip89], and Blum, Luby and Rubinfeld [BLR90]), as well as some new ideas about how to represent logical formulae with polynomials (Babai and Fortnow [BF91], Fortnow, Lund, Karloff and Nisan [LFKN92], and Shamir [Sha92]) were used to show that IP= PSPACE([LFKN92, Sha92]) and MIP= NEXPTIME(Babai, Fortnow, and Lund[BFL91]). These connections between traditional and nontraditional complexity classes were proved using novel algebraic technques, some of which will be covered later in this book.

Emergence of PCP. The characterization of MIP from ([FRS88]) and the result MIP= NEXPTIME together imply that NEXPTIME is exactly the set of languages for which membership proofs can be checked by a probabilistic polynomial time verifier. Such a surprising result led to some thinking about NP as well, specifically, the papers of Babai, Fortnow, Levin, and Szegedy ([BFLS91]) and Feige, Goldwasser, Lovász, Safra, and Szegedy ([FGL$^+$91]). Although only the latter talked explicitly about NP (the former dealt with checking nondeterministic computations; including NP computations as a subcase), their techniques were actually scaling down the MIP=NEXPTIME result. The paper [FGL$^+$91] implicitly defined a hierarchy of complexity classes – unnamed there, but which we can call C. Their class $C(t(n))$ is identical to the class PCP$(t(n), t(n))$ as defined in this chapter, and their main result was that NP $\subseteq C(\log n \cdot \log\log n)$.

What caused great interest in [FGL$^+$91] was their corollary: If the clique number of a graph can be approximated within any fixed constant factor, then all NP problems can be solved deterministically in time $n^{O(\log\log n)}$. (Computing the clique number is a well-known NP-complete problem [Kar72].) They showed how to reduce every problem in PCP$(\log n \cdot \log\log n)$ – and as a special subcase, every problem in NP – to the Clique problem in such a way that the "gap" (probability 1 versus probability 1/2) used in the definition of PCP translates into a gap in the clique number.

To prove that NP $\subseteq C(\log n)$ seemed to be the next logical step, and for two reasons. First, this would imply that approximating the clique number is NP-hard. Second, since $C(\log n)$ is trivially a subclass of NP, such a result would imply a new characterization of NP, namely, NP $= C(\log n)$.

This step was taken in the paper (Arora and Safra [AS92]). Somewhat curiously, we found that although the number of random bits used by the verifier cannot be sub-logarithmic (or else P = NP; see [AS92]), there was no such restriction on the number of query-bits. Hence we defined the class PCP (definition 2.5) with two parameters (instead of the single parameter used in [FGL+91]). We showed that NP = $\mathrm{PCP}(\log n, (\log\log n)^2)$.

At this point, several people (for example, the authors of [AMS+92]) realized that proving NP = $\mathrm{PCP}(\log n, 1)$ would prove the inapproximability of MAX-3SAT. This result was actually obtained in the paper (Arora, Lund, Motwani, Sudan and Szegedy [ALM+92]). Owing to its great dependence upon [AS92], the proof of the PCP Theorem is often attributed to jointly to [ALM+92, AS92].

Among other papers that were influential in the above developments were those by Beaver and Feigenbaum ([BF90]), Lapidot and Shamir ([LS91]), Rubinfeld and Sudan [RS92], and Feige and Lovasz ([FL92]). Their contributions will be described in appropriate places later.

Other characterizations of NP. Researchers have discovered other probabilistic characterizations of NP. One such result, implicit in Lipton's paper ([Lip89]), says that NP is exactly the set of languages for which membership proofs can be checked by a probabilistic logspace verifier that uses $O(\log n)$ random bits, and makes just one sweep (say left-to-right) over the proof-string. Condon and Ladner ([CL89]) further strengthened Lipton's characterization. Even more interestingly, Condon ([Con93]) then used the result of ([CL89]) to show the hardness of approximating the max-word problem. This unapproximability result appeared somewhat before (and was independent of) the more well-known [FGL+91] paper.

An older characterization of NP, in terms of spectra of second-order formulae, is due to Fagin([Fag74]). His result, since it involves no notion of computation, is an interesting alternative viewpoint of NP. It has motivated the definition of many classes of approximation problems, including MAX-SNP.

(*The above account of the evolution of PCP has been kept brief; for more details refer to the surveys by Babai ([Bab94]) and Johnson ([Joh92]). A recent survey by Goldreich ([Gol94]) describes the known variations on probabilistic proof systems, and how they are used in cryptography.*

Three existing dissertations describe various aspects of the above theory. Rubinfeld's ([Rub90]) describes the theory of program checking, Lund's ([Lun92]) describes the surprising results on interactive proofs, and Sudan's ([Sud92]) describes program checking for algebraic programs and its connection to the PCP Theorem.)

Chapter 3

PCP : An Overview

The PCP theorem (Theorem 2.1) states merely the existence of a verifier for any NP problem. Our proof of the theorem is quite constructive: we give an explicit program for the verifier, as well as explicit descriptions of what the proof must contain in order to be accepted with probability 1.

Details come later, but let us first face a fundamental problem to be solved: How can the verifier recognize the proof as valid or invalid, after examining only $O(1)$ bits in it? At the very least, an invalid proof must differ from a valid one on a large fraction of bits, so that they appear different to a randomized test that examines very few bits in them. This consideration alone suggests using the theory of error-correcting *codes*. Although we do not need much of the classical machinery of coding theory, some of its terminology is very useful.

3.1. Codes

Let Σ be an alphabet of symbols, and m an integer. Let a *word* be a string in Σ^m. The *distance* between two words is the fraction of coordinates in which they differ. (This distance function is the well-known *Hamming* metric, but scaled to lie in $[0,1]$.) For $\tau \in [0,1]$ let $\mathrm{Ball}(y,\tau)$ denote the set of words whose distance to y is less than τ.

A *code* $\mathcal{C}$ is a subset of Σ^m; every word in $\mathcal{C}$ is called a *codeword*. A word y is *γ-close to* $\mathcal{C}$ if there is a codeword in $\mathrm{Ball}(y,\gamma)$. (We say just γ-close when $\mathcal{C}$ is understood from the context.)

The *minimum distance* of $\mathcal{C}$, denoted $\delta_{\min}$, is the minimum distance between two codewords. Note that if word y is $\delta_{\min}/2$-close, there is exactly one codeword z in $\mathrm{Ball}(y,\delta_{\min}/2)$ (for, if z' is another such codeword, then by the triangle inequality, $\delta(z,z') \le \delta(z,y) + \delta(y,z') < \delta_{\min}$, which is a contradiction). Let $\mathrm{N}(y)$ denote this nearest codeword to y.

A code $\mathcal{C}$ is useful to us as an encoding object: using it we can encode strings of bits.

For any integer k such that $2^k \leq |\mathcal{C}|$, let σ be a one-to-one map from $\{0,1\}^k$ to $\mathcal{C}$ (such a map clearly exists; in our applications it will be defined in an explicit fashion). For $x \in \{0,1\}^k$, we call the codeword $\sigma(x)$ an *encoding* of x. Note that $\forall x, y \in \{0,1\}^k$, we have $\delta(\sigma(x), \sigma(y)) \geq \delta_{\min}$. We emphasize that σ need not be onto, that is, σ^{-1} is not defined for all codewords.

Example 3.1: Let F be the field GF(q), and h an integer less than $q/3$. Consider F as an alphabet and define a code $\mathcal{C}_1 \subset \mathrm{F}^{3h}$ as the following set (of size $|\mathrm{F}|^{h+1}$.

$$\left\{ (\sum_{i=0}^{h} a_i b_1^i, \sum_{i=0}^{h} a_i b_2^i, \ldots, \sum_{i=0}^{h} a_i b_{3h}^i) : \quad a_0, \ldots, a_h \in \mathrm{F} \right\}$$

where $b_1, \ldots, b_{3h}$ are distinct elements of F. Note that a codeword is the sequence of values taken by some polynomial $\sum_{i=0}^{h} a_i x^i$ at these $3h$ points. Since two distinct polynomials of degree h agree on at most h points, the minimum distance between any two codewords in $\mathcal{C}_1$ is at at least $2h/3h = 2/3$.

What is the alphabet size, q $(= |\mathrm{F}|)$, as a function of the code-size, $|\mathcal{C}_1|$? Let N denote the number of codewords, that is, q^{h+1}. We assumed that $q > 3h$, so we have $(q)^{q/3} > N$. Hence $q = \Omega(\frac{\log N}{\log \log N})$.

Here's one way to define $\sigma : \{0,1\}^{h+1} \to \mathrm{F}^{3h}$. For $a_0, \ldots, a_h \in \{0,1\}$ define

$$\sigma(a_0, \ldots, a_h) = (\sum_{i=0}^{h} a_i b_1^i, \sum_{i=0}^{h} a_i b_2^i, \ldots, \sum_{i=0}^{h} a_i b_{3h}^i).$$

3.2. Proof of the PCP Theorem: an Outline

We know of no simple, direct proof of the PCP Theorem (Theorem 2.1). The only known proof – the one presented here – uses 2 different verifiers which are combined in a hierarchical construction using Lemma 3.1. There is a trade-off between the number of random-bits and query-bits used by the two verifiers, which the construction exploits. To enable this construction we require the verifiers to be in a certain *normal form*, which is described below. Moreover, we associate a new parameter with a verifier namely, *decision time*, which is the chief parameter of interest in the hierarchical construction.

Recall (from the description before Definition 2.4) that a verifier's operation may be viewed as having three stages. The first stage reads the input and the random string, and decides what locations to examine in Π. The second stage reads symbols from Π onto the work-tape. The third stage decides whether or not to accept.

Definition 3.1: The *decision time* of a verifier is the time taken by the third stage.

Next, we describe the normal form. For ease we describe verifiers for the language 3SAT. (Since 3SAT is NP-complete, verifiers for other NP languages are trivial modifications of the verifier for 3SAT.) Denote by φ the input 3CNF formula, and by n the number of its variables. Identify, in the obvious way, the set of strings in $\{0,1\}^n$ with the set of assignments to variables of φ. Finally, let Π denote the provided proof-string.

Definition 3.2: A verifier for 3SAT is in *normal form* if it satisfies the following properties.

1. **Has a certain alphabet.** The verifier expects the proof to be a string over a certain alphabet, say Σ (the size of Σ may depend upon the input size n). A query of a verifier involves reading a symbol of Σ, and not just a bit.

2. **Can check assignments that are split into many parts.** The verifier has a special subroutine for efficiently checking proofs of a very special form. Let p be any given positive integer. The subroutine behaves as follows.

   (i) It defines an associated code $\mathcal{C}$ over the alphabet Σ, with $\delta_{\min} > 0.3$. The code has an associated one-to-one map σ from strings in $\{0,1\}^{\frac{n}{p}}$ to codewords.

   (ii) It expects Π to have the special form

   $$\sigma(a_1) \circ \sigma(a_2) \circ \cdots \circ \sigma(a_p) \circ \pi$$

   ($\circ$ = concatenation of strings), where π is a string that is supposed to show that the string $a_1 \circ \cdots \circ a_p$ is a satisfying assignment.

   More formally, we say that the subroutine can *check assignments split into p parts* if it has the following behavior on proofs of the form $z_1 \circ \cdots \circ z_p \circ \pi$, where z_i's and π are strings over alphabet Σ.

   - If $z_1, \ldots, z_p$ are codewords such that $\sigma^{-1}(z_1) \circ \cdots \sigma^{-1}(z_p)$ is a satisfying assignment, then there is a π such that

     $$\Pr[\text{ subroutine accepts } z_1 \circ \cdots \circ z_p \circ \pi] = 1.$$

   - If $\exists i:\ 1 \le i \le p$ such that z_i is not $\delta_{\min}/3$-close, then for all π,

     $$\Pr[\text{ subroutine accepts } z_1 \circ \cdots \circ z_p \circ \pi] < \frac{1}{2}.$$

   - If all z_i's are $\delta_{\min}/3$-close, but $\sigma^{-1}(N(z_1)) \circ \cdots \circ \sigma^{-1}(N(z_p))$ is not a satisfying assignment, where $N(z_i)$ is codeword nearest to z_i, then again for all π

     $$\Pr[\text{ subroutine accepts } z_1 \circ \cdots \circ z_p \circ \pi] < \frac{1}{2}.$$

Now we modify Definition 2.3 to make it meaningful for a verifier in normal form.

Definition 3.3: A verifier in normal form is $(r(n), q(n), t(n))$-*restricted* if on inputs of size n it uses $O(r(n))$ random bits, reads $O(q(n))$ symbols from Π, and has decision time $\text{poly}(t(n))$. While checking assignments split into p parts, it reads $O(p \cdot q(n))$ symbols.

Note: The parameter $q(n)$ refers to the number of symbols (that is, elements of the alphabet Σ), read from Π. Thus the number of *bits* of information read from the proof is $O(q(n) \log |\Sigma|)$. We choose not to make $|\Sigma|$ a parameter, since there is already an implicit upperbound for it in terms of the above parameters. Realize that the decision time includes the time to process $O(q(n) \log |\Sigma|)$ bits of information, so $O(q(n) \log |\Sigma|) \leq \text{poly}(t(n))$. This upperbound is good enough for our purposes.

Now we describe a general technique to reduce the decision time, and in the process, the number of bits of information read from the proof.

Lemma 3.1 (Composition Lemma): *Let V_1 and V_2 be normal-form verifiers for 3SAT that are $(R(n), Q(n), T(n))$-restricted and $(r(n), q(n), t(n))$-restricted respectively. Then there is normal form 3SAT verifier that is $(R(n)+r'(n),\ Q(n) \cdot q'(n),\ t'(n))$-restricted, where $r'(n) = r(\mathit{poly}(T(n)))$, $q'(n) = q(\mathit{poly}(T(n)))$, and $t'(n) = t(\mathit{poly}(T(n)))$.*

Note: Whenever we use this lemma, both $Q(n), q(n)$ are $O(1)$. Then the three verifiers respect the bounds $(R(n), 1, T(n))$, $(r(n), 1, t(n))$, and $(R(n)+r(\text{poly}(T(n))), 1,\ t(\text{poly}(T(n)))$ respectively. Think of t as being some slowly-growing function like log. Then the decision time of the new verifier is at most $\log(\text{poly}(T(n))) = O(\log(T(n))$, an exponential improvement over $T(n)$.

We will prove the Composition Lemma at the end of this section. First we outline how it is used to prove the PCP Theorem. The essential ingredients are Theorem 3.2 and 3.4, which will be proved in Chapter 4.

Theorem 3.2: *3SAT has a $(\log n, 1, \log n)$-restricted normal form verifier.*

Although the above verifier reads only $O(1)$ symbols in the proof, the number of bits it reads is $\text{poly}(\log n)$. We use the Composition Lemma to improve it.

Corollary 3.3: *3SAT has a normal-form verifier that is $(\log n, 1, \log\log n)$-restricted.*

Proof: Use the verifier of theorem 3.2 to play the role of both V_1 and V_2 in the Composition Lemma. $\Box$

Since the verifier of Corollary 3.3 is in normal form we could apply the Composition Lemma again (the reader may wish to calculate the best PCP result obtained this way). Instead we use a new verifier to terminate the composition. It requires a huge number of random bits, but is really efficient with its queries.

Theorem 3.4: *3SAT has a normal-form verifier that is* $(n^3, 1, 1)$*-restricted, and uses the alphabet* $\{0, 1\}$.

Now we can prove the following theorem, a strong form of the PCP theorem.

Theorem 3.5: *There exists a normal form verifier for 3SAT that is* $(\log n, 1, 1)$*-restricted, and uses the alphabet* $\{0, 1\}$.

Proof: Use the verifiers of corollary 3.3 and theorem 3.4 as V_1 and V_2 respectively in the Composition Lemma. □

The verifier of Theorem 3.5, like all our verifiers, is a verifier for 3SAT. By examining its performance not only do we conclude that NP = PCP$(\log n, 1)$, but also that the verifier in question is in normal form. (For this and other strong forms of the PCP theorem, see Chapter 8).

Our outline of the proof of the PCP Theorem is complete. We have shown that it suffices to prove the Composition Lemma, which we will do now, and Theorems 3.2 and 3.4, which we will do in chapter 4.

Proof: (*Of Composition Lemma*) The ideas underlying the composition are simple. Once we fix the random string of the first verifier V_1, its decision depends upon a very small portion of the proof string, and is computed in very little time (namely, the decision time). The Cook-Levin Theorem implies that a tiny 3CNF formula describes whether the decision is an accept or a reject. We modify verifier V_1 to use verifier V_2 - which is in normal form – to check that the above-mentioned portion of the proof is a satisfying assignment to this tiny 3SAT formula. Doing this involves using V_2's ability to check split assignments, and requires that relevant portions of the proof-string be present in an encoded form (using V_2's encoding).

Now we provide details. Let n be the size of the input given to verifier V_1, and let Q, R, T denote, respectively, the number of queries made by the verifier, the number of random bits it uses, and its decision time. (The hypothesis of the lemma implies that Q, R, T are $O(Q(n)$, $O(R(n))$ and poly$(T(n))$ respectively.)

Let the random string of verifier V_1 be fixed to $r \in \{0, 1\}^R$. This fixes the sequence of locations in proof Π that the verifier will examine. Let $i_1, i_2, \ldots, i_Q$ denote these locations. (Strictly speaking, we should express the dependence upon r explicitly and denote these by $i_1(r), i_2(r), \ldots,$ etc.) The decision process of V_1 is a computation that runs in time T using an input $\Pi[i_1] \circ \Pi[i_2] \circ \cdots \circ \Pi[i_Q]$, where $\Pi[j]$ denotes the symbol in the jth location of proof Π, and $\circ$ denotes concatenation of strings. The strong form of the Cook-Levin theorem (Fact A.1) implies we can write down a 3SAT formula ψ_r of size poly(T) such that V_1 accepts iff

$$\exists y_r \;\; \text{s.t. } \Pi[i_1] \circ \Pi[i_2] \circ \cdots \circ \Pi[i_Q] \circ y_r \;\; \text{is a satisfying assignment for } \psi_r. \tag{3.1}$$

(Here we are thinking of each symbol $\Pi[i_j]$ as being represented by a string of bits.)

To make our description cleaner, we assume from now on that Π contains a sequence of additional locations, one for each choice of the random string r. The rth location supposedly contains y_r. Further, we assume that V_1 when using the random string r makes a separate query to this location to read y_r. Let i_{Q+1} be the index of this location. Then V_1 accepts using the random string r iff

$$\Pi[i_1] \circ \Pi[i_2] \circ \cdots \circ \Pi[i_{Q+1}] \text{ is a satisfying assignment for } \psi_r. \tag{3.2}$$

But there is a very efficient way to determine the truth of a statement like the one in (3.2): Use V_2's subroutine for checking assignments split into $Q+1$ parts. Of course, this requires the structure of Π to be somewhat different. The symbol in each location of Π is now required to be encoded using V_2's map σ from bit-strings to codewords.

The next page contains the program of V_3, the new verifier obtained by the composition. It uses V_2 to check assignments for ψ_r that are split into $Q+1$ parts. (We assume — without loss of generality, by "padding" with irrelevant bits — that the string y_r in (3.1) has the same length as each $\Pi[j]$. So these $Q+1$ parts have equal size.) Let m denote the number of variables in ψ_r. (Note: $m = \text{poly}(T)$.) Let σ denote V_2's map from bit-strings of size $m/(Q+1)$ to codewords, and let R' be the number of random bits it uses while checking the split assignment. (Note: By the hypothesis of the lemma, $R' = O(r(m))$.) For convenience, we assume (by repeating the program of V_2 some $O(\log 1/\epsilon)$ times) that when the input formula is not satisfiable, V_2 rejects with probability at least $1-\epsilon$, instead of just $1/2$ as required by definition of a verifier (ϵ is a small enough constant, say 0.1).

Complexity: We analyze V_3's efficiency. Let m denote $|\psi_r|$. The verifier uses $R + R' = R + r(m)$ random bits, which is $O(R(n) + r(\text{poly}(T(n))))$. Further, the number of queries it makes and its decision time are exactly those of V_2's subroutine. Since V_2 is $(r(n), q(n), t(n))$-restricted, its subroutine makes $(Q+1)\cdot q(m)$ queries while checking assignments split into $Q+1$ parts, and has decision time $t(m)$. We conclude that V_3 is $(R + r(m), (Q+1)\cdot q(m), t(m))$-restricted. But $m = |\psi_r| = \text{poly}(T(n))$, so the parameters for V_3 are as claimed.

Program of V_3.

Given: Table Π_{new} with the same number of locations as Π.
Table Γ with 2^R entries.

Pick a random $r \in \{0,1\}^R$ and a random $r' \in \{0,1\}^{R'}$.
Use V_1 to generate locations $i_1(r), \ldots, i_{Q+1}(r)$, and 3SAT formula ψ_r.
Run V_2's subroutine (using random string r') to check that the
proof $z_1 \circ \cdots \circ z_{Q+1} \circ \rho_r$ encodes a satisfying assignment for ψ_r,
where z_j is the entry $\Pi_{new}[i_j(r)]$ and ρ_r the entry $\Gamma[r]$.
ACCEPT iff V_2's subroutine accepts.

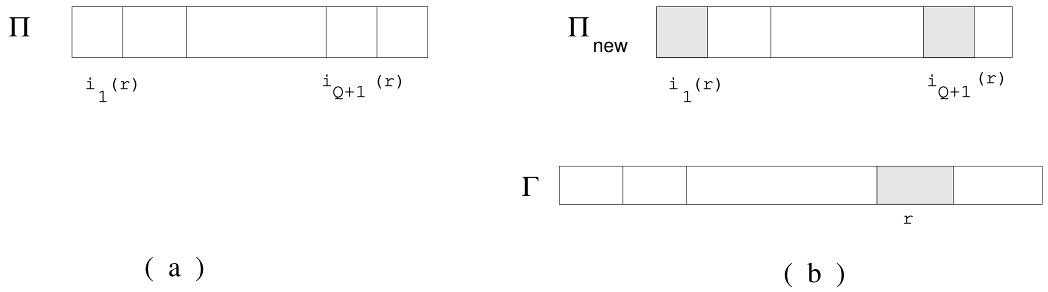

Figure 3.1: (a) Proof that V_1 expects. (b) Proof that V_3 expects. Shaded area in Π_{new} represents the assignment split into $Q+1$ parts that corresponds to V_1's random seed r.

Correctness: Showing that V_3 is a correct verifier for 3SAT consists in two parts. In both parts, let a_r denote the string $\Pi[i_1(r)] \circ \cdots \circ \Pi[i_{Q+1}(r)$ and s_r denote the string $\sigma(\Pi[i_1(r)]) \circ \cdots \sigma(\Pi[i_{Q+1}(r)])$. It will be clear from the context what Π refers to.

First, suppose the input φ is satisfiable. Then there is a proof Π that V_1 accepts with probability 1. We show that there exists a proof (Π_{new}, Γ) which V_3 accepts with probability 1.

Since V_1 accepts Π with probability 1, we have

$$\Pr_{r \in \{0,1\}^R} [\ a_r \text{ is a satisfying assignment for } \psi_r] = 1$$

where ψ_r is the 3SAT formula representing the verifier's computation. Definition 3.2 about what it means to "check assignments split into $Q+1$ parts" implies that for every $r \in \{0,1\}^R$ there is a string π_r such that

$$\Pr_{r' \in \{0,1\}^{R'}} [V_2\text{'s subroutine accepts } s_r \circ \pi_r \text{ using } r' \text{ as a random string}] = 1. \tag{3.3}$$

Construct the desired new proof (Π_{new}, Γ) as follows. Let Π_{new} be a table with the same number of locations as Π, whose jth location contains the codeword $\sigma(\Pi[j])$. Let Γ be a table of 2^R locations, viewed as being in one-to-one correspondence with $\{0,1\}^R$. For $r \in \{0,1\}^R$, let the location r of Γ contain the string π_r defined in Equation (3.3). The decision of the new verifier V_3 when it picks r from $\{0,1\}^R$, and r' from $\{0,1\}^{R'}$ is just the decision of V_2's subroutine on the proof-string $s_r \circ \pi_r$. Our construction insures that the subroutine accepts irrespective of r, r'. Hence

$$\Pr_{r,r'} [V_3 \text{ accepts } (\Pi_{new}, \Gamma)] = 1.$$

Now we turn to the second half: φ is unsatisfiable. Let (Π_{new}, Γ) be any proof and let p be the probability with which V_3 accepts it. We show that $p < 1/2 + 2\epsilon$.

First, modify the proof-string as follows: replace each entry of Π_{new} by the codeword nearest to it (or *a* codeword nearest to it, if there is more than one choice). Call this new table Π'_{new}. Then V_3 must accept (Π'_{new}, Γ) with probability at least $p - \epsilon$. For, V_3 accepts iff V_2's subroutine for checking split assignments accepts. The subroutine accepts with probability no more than ϵ if even one of the parts in (the encoding of) the split assignment is not $\delta_{\min}$-close. Hence turning the entries of Π_{new} into codewords as above can lower the probability of acceptance by at most ϵ.

Next, we turn Π'_{new} into a proof that the original verifier V_1 accepts with almost the same probability as V_3 does. Define Π by making its jth entry the preimage of the jth entry of Π'_{new}, that is, $\Pi[j] = \sigma^{-1}(\Pi'_{new}[j])$ (note: if $\sigma^{-1}(\Pi'_{new})$ is not defined, use an arbitrary symbol instead).

Let γ be the probability with which V_1 rejects Π. (Since φ is unsatisfiable, γ is more than $1/2$.) That is to say, γ is the fraction of $r \in \{0,1\}^R$ for which

$$\Pr_{r\in\{0,1\}^R}[a_r \text{ does not satisfy } \psi_r] = \gamma.$$

Let r be one of the choices of the random string for which a_r does not satisfy ψ_r. That is, $\Pi[i_1(r)] \circ \cdots \circ \Pi[i_{Q+1}(r)]$ does not satisfy ψ_r. Since s_r is $\sigma(\Pi[i_1(r)]) \circ \cdots \sigma(\Pi[i_{Q+1}(r)])$, we have

$$\Pr_{r'\in\{0,1\}^{R'}}[V_2\text{'s subroutine accepts } s_r \circ \pi \text{ using the random string } r'] \leq \epsilon,$$

irrespective of π. In particular, V_2's subroutine accepts the proof $s_r \circ \Gamma[r]$ with probability at most ϵ. Hence we have

$$\Pr_{r,r'}[V_3 \text{ accepts } (\Pi'_{new}, \Gamma)] \leq \gamma\epsilon + (1-\gamma).$$

But we assumed that V_3 accepts Π'_{new}, Γ with probability is at least $p - \epsilon$. Therefore we have

$$p - \epsilon \leq \frac{1}{2}\epsilon + \frac{1}{2},$$

which implies $p < 1/2 + 2\epsilon$. Thus the claim of correctness is proved. □

3.3. History and Background

The first PCP-type verifier appears in the result MIP = NEXPTIME [BFL91]. When Babai et al. ([BFLS91]) scaled down that result to NP, they noticed that their PCP-type verifier for NP has a low decision time. (Actually their exact result was somewhat stronger.) This observation motivated the work in ([AS92]), where the Composition Lemma is implicit. The use of large-distance codes in the definition of the normal form verifier was also motivated by a similar situation in [BFLS91]. Composition was termed "Recursion" in [AS92] because in that paper verifiers were composed only with themselves. Full use of the lemma (as described in this section) was made in [ALM+92], where Theorems 3.2 and 3.4 were proven. (The best verifier of [AS92] was $(\log n, (\log\log n)^2, (\log\log n)^2)$-restricted.) The idea of composing verifiers has been crucial in more recent developments in the area of probabilistically checkable proofs (specifically, in constructing more efficient PCPs).

Chapter 4

A Proof of the PCP Theorem

This chapter contains a proof of the PCP theorem. As per the outline in Chapter 3, it suffices to construct the verifiers of Theorems 3.2 and 3.4. First we give a brief overview of the techniques used.

Underlying the description of both verifiers is an algebraic representation of a 3SAT formula. The representation uses a simple fact: every assignment can be encoded as a multivariate polynomial that takes values in a finite field (see Section 4.1). A polynomial that encodes a satisfying assignment is called a *satisfying polynomial*. Just as a satisfying boolean assignment can be recognized by checking whether or not it makes all the clauses of the 3SAT formula true, a satisfying polynomial can be recognized by checking whether it satisfies some set of equations involving the operations $+$ and $\cdot$ of the finite field.

Each of our verifiers expects the proof to contain a polynomial, plus some additional information showing that this polynomial is a satisfying polynomial (in other words, it satisfies the above-mentioned set of equations). The verifier checks this information using some algebraic procedures connected with polynomials. These procedures are described in a "black-box" fashion in Section 4.1.1, and in full detail in Section 4.4. The black-box description should suffice to understand Sections 4.2 and 4.3, in which we prove Theorems 3.4 and 3.2 respectively.

All results in this chapter are self-contained, except Theorem 4.15 about the performance of the low-degree test, whose proof takes up Chapter 5.

Throughout the chapter φ denotes the 3SAT formula for which proofs are being checked. We use n to denote both the number of clauses and the number of variables. (We defend this usage on the grounds that the number of variables and clauses could be made equal by adding irrelevant variables – which do not appear in any clause – to the set of variables.)

A note on error probabilities. While describing verifiers we write in parenthesis and in *italics* the conditions that a proof must satisfy in the good case (i.e., the case when φ is satisfiable). The reader may wish to check that there exists a proof meeting those conditions, and which is therefore accepted with probability 1. Upperbounding the probability of rejection in the bad case – when φ is not satisfiable – by 1/2 is more difficult, and requires proof.

The overall picture. After reading the proof, a look at Figure 4.3 might help the reader recall all important steps.

4.1. Polynomial Codes and Their Use

Let F be the finite field GF(q) and k, d be integers. A *k-variate polynomial of degree d over F* is a sum of terms of the form $ax_1^{j_1}x_2^{j_2}\cdots x_k^{j_k}$ where $a \in \mathrm{F}$ and integers $j_1, \ldots, j_k$ satisfy $j_1 + \cdots + j_k \leq d$. Let $\mathrm{F}_d[x_1, \ldots, x_k]$ be the set of functions from F^k to F that can be described by a polynomial of total degree at most d. [1]

We will be interested in representations of polynomials *by value.* A k-variate polynomial defines a function from F^k to F, so it can be expressed by $|\mathrm{F}|^k = q^k$ values. In this representation a k-variate polynomial (or any function from F^k to F for that matter) is a word of length q^k over the alphabet F.

Definition 4.1: The *code of k-variate polynomials of degree d* (or just *polynomial code* when k, d are understood from context) is the code $\mathrm{F}_d[x_1, \ldots, x_k]$ in F^{q^k}.

Note that the distance between two words is the fraction of points in F^k they disagree on. The following lemma (for a proof see Appendix A) shows that the polynomial code has large minimum distance.

Fact 4.1 (Schwartz): *Two distinct polynomials in $F_d[x_1, \ldots, x_k]$ disagree on at least $1 - d/q$ fraction of points in F^k.* $\Box$

In our applications, $d < q/2$. Thus if $f : \mathrm{F}^k \to \mathrm{F}$ is a δ-close function ($\delta < 1/4$, say) then the polynomial that agrees with it in at least $1 - \delta$ fraction of the points is unique. (In fact, no other polynomial describes f in more than even $\delta + d/q$ fraction of the points.) We will often let δ denote a suitably less constant, say 0.1.

Definition 4.2: For a 0.1-close function f the unique nearest polynomial is denoted by $\widetilde{f}$.

Polynomials are useful to us as encoding objects. We define below a canonical way (due to [BFLS91]) to encode a sequence of bits with a polynomial. For convenience we describe a more general method that encodes a sequence of field elements with a polynomial. Encoding a sequence of bits is a sub-case of this method, since $0, 1 \in \mathrm{F}$.

[1] The use of F_d above should not be confused with the practice in some algebra texts of using F_q as a shorthand for GF(q).

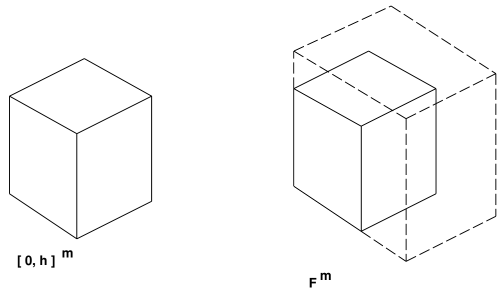

Figure 4.1: A function from $[0,h]^m$ to F can be extended to a polynomial of degree mh.

Let h be an integer such that the set of integers $[0,h]$ is a subset of the field F. (Readers uncomfortable with this notation can think of $[0,h]$ as any subset of the field F that has size $h+1$).

Theorem 4.2: *For every function $s : [0,h]^m \to F$ there is a function $\widehat{s} \in F_{mh}[x_1, \ldots, x_m]$ such that $s(y) = \widehat{s}(y)$ for all $y \in [0,h]^m$.*

Proof: For $\overline{u} = (u_1, \ldots, u_m) \in [0,h]^m$ let $L_{\overline{u}}$ be the polynomial defined as

$$L_{\overline{u}}(x_1, \ldots, x_m) = \prod_{i=1}^{m} l_{u_i}(x_i),$$

where l_{u_i} is the unique degree-h polynomial in x_i that is 1 at $x_i = u_i$ and 0 at $x_i \in [0,h] \setminus \{u_i\}$. (That $l_{u_i}(x_i)$ exists follows from Fact A.3.) Note that the value of $L_{\overline{u}}$ is 1 at $\overline{u}$ and 0 at all the other points in $[0,h]^m$. Also, its degree is mh.

Now define the polynomial $\widehat{s}$ as

$$\widehat{s}(x_1, \ldots, x_m) = \sum_{\overline{u} \in [0,h]^m} s(\overline{u}) \cdot L_{\overline{u}}(x_1, \ldots, x_m).$$

□

Example 4.1: Let $m = 2, h = 1$. Given any function $f : [0,1]^2 \to$ F we can map it to a bivariate degree 2 polynomial, $\widehat{f}$, as follows.

$$\begin{aligned} \widehat{f}(x_1, x_2) &= (1 - x_1)(1 - x_2) f(0,0) + x_1(1 - x_2) f(1,0) \\ &\quad + (1 - x_1) x_2 f(0,1) + x_1 x_2 f(1,1). \end{aligned}$$

Definition 4.3: For a function $s : [0,h]^m \to$ F, a *polynomial extension of* s is a function $\widehat{s} \in \mathrm{F}_{mh}[x_1, \ldots, x_m]$ that satisfies

$$\widehat{s}(y) = s(y) \quad \forall y \in [0,h]^m.$$

(note: the extension need not be unique).

The encoding. Let h be an integer such that $[0,h] \subseteq \mathrm{F}$, and l an integer such that $l = (h+1)^m$ for some integer m. Define a one-to-one map from F^l to $\mathrm{F}_{mh}[x_1,\ldots,x_m]$ (in other words, from sequences of l field elements to polynomials in $\mathrm{F}_{mh}[x_1,\ldots,x_m]$) as follows. Identify in some canonical way the set of integers $\{1,\ldots,l\}$ and the set $[0,h]^m \subseteq \mathrm{F}^m$. (For instance, identify the integer $i \in \{1,\ldots,l\}$ with its m-digit representation in base $h+1$.) Thus a sequence s of l field elements may be viewed as a function s from $[0,h]^m$ to F. Map the sequence s to any polynomial extension $\widehat{s}$ of this function. This map is one-to-one because if polynomials $\widehat{f}$ and $\widehat{g}$ are the same, then they agree everywhere and, in particular, on $[0,h]^m$, which implies $f = g$.

The inverse map of the above encoding is obvious. A polynomial $f \in \mathrm{F}_{mh}[x_1,\ldots,x_m]$ is the polynomial extension of the function $r : [0,h]^m \to \mathrm{F}$ defined as $r(x) = f(x), \forall x \in [0,h]^m$.

Note that we encode sequences of length $l = (h+1)^m$ by sequences of length $|\mathrm{F}|^m = q^m$. In later applications this increase in size is not too much. The applications depend upon some algebraic procedures to work correctly, for which it suffices to take $q = \mathrm{poly}(h)$. Then q^m is $h^{O(m)} = \mathrm{poly}(l)$. Hence the increase in size is polynomially bounded.

Next, we indicate how we will use the above encoding.

Definition 4.4: Let φ be a 3CNF formula with n variables and F be a field. A sequence of field elements $a_1, a_2, \ldots, a_n \in \mathrm{F}$ is said to *represent a satisfying assignment* if the following (partial) boolean assignment makes all clauses in φ true: If $a_i = 0$ (resp., $a_i = 1$), let the ith variable be false (resp., true) and otherwise do not assign a value to the ith variable.

A key concept connected with our verifiers is that of a *satisfying polynomial.*

Definition 4.5: Let φ be a 3CNF formula with n variables and F be a field. A *satisfying polynomial of* φ is a polynomial extension (for any appropriate choice of parameters m and h as above) of a sequence of field elements that represents a satisfying assignment to φ.

(As noted above, when the parameters $m, h, |\mathrm{F}|$ are chosen appropriately, the satisfying polynomial can be represented by $\mathrm{poly}(n)$ bits.)

Representing clauses by equations. We give a simple algebraic condition to characterize sequences of field elements that represent satisfying assignments. (Our verifiers will use these conditions to check whether or not a given polynomial is a satisfying polynomial.)

Lemma 4.3: *Let φ be a 3SAT instance with n variables and n clauses. Let F be any field and let $X_1,\ldots,X_n$ be formal variables taking values over F. There is a set of n cubic equations $\{p_i(X_{i_1}, X_{i_2}, X_{i_3}) = 0 : i = 1,..,n\}$, such that a sequence $a_1,\ldots,a_n \in F$ represents a satisfying assignment iff*

$$p_i(a_{i_1}, a_{i_2}, a_{i_3}) = 0 \quad \forall i \in [1,n] \tag{4.1}$$

The set of equations can be constructed in poly(n) time.

Proof: Let $y_1, \ldots, y_n$ be the variables of φ. *Arithmetize* each clause of φ as follows. For $i = 1, \ldots, n$, associate the field variable X_i with the boolean variable y_i. In each clause replace y_i by $1 - X_i$, the operation $\vee$ by $\cdot$ (= multiplication over F) and $\neg y_i$ by X_i. Thus for example, clause $y_i \vee \neg y_j \vee y_k$ is replaced by the cubic equation $(1 - X_i) \cdot X_j \cdot (1 - X_k) = 0$. Note that the expression on the left hand side of this equation is is 0 iff $X_i = 1$, or $X_j = 0$, or $X_k = 1$. That is to say, values of X_i, X_j, X_k that satisfy the equation correspond in a natural way to boolean values of y_i, y_j, y_k that make the clause true.

If the variables involved in the ith clause are $y_{i_1}, y_{i_2}, y_{i_3}$, the arithmetization above yields a cubic equation $p_i(X_{i_1}, X_{i_2}, X_{i_3}) = 0$ for this clause. Thus we get n equations, one per clause, such that the assignment $X_1 = a_1, \ldots X_n = a_n$ satisfies all of them iff the corresponding boolean assignment to $y_1, \ldots, y_n$ satisfies φ. $\Box$

4.1.1. Algebraic Procedures for Polynomial Codes

In this section we give a "black-box" description of some algebraic procedures concerning polynomial codes (for details of how they work refer to Section 4.4). First we explain how a verifier uses them.

The verifier defines (using the polynomial extension encoding from Definition 4.3) a mapping from boolean assignments to polynomials of degree d for some suitable d, and expects the proof to contain one such polynomial. Recall that polynomials are represented by value, so the proof actually contains some table of values $f : \mathrm{F}^m \to \mathrm{F}$. How can the verifier check that $f \in \mathrm{F}_d[x_1, \ldots, x_m]$? Our first procedure, the test for δ-closeness, allows it to do almost that. By looking at very few values of f the procedure determines whether or not f is δ-close (where δ is some suitably small constant). So suppose f is indeed found to be δ-close. Our second procedure can reconstruct values of $\widetilde{f}$, the polynomial closest to f, at any desired points. Together, the two procedures allow the verifier to assume for all practical purposes that f is exactly a polynomial.

Actually we describe two pairs of procedures. The first pair is somewhat specialized, and works only for a special polynomial code called the *linear function code*, which is the code $\mathrm{F}_d[x_1, \ldots, x_m]$ where F is GF(2), the field of two elements, and degree d is 1. This code will be used in Section 4.3.

Procedure 4.1 (Procedures for the linear function code.): Let F be the field GF(2).

(i) Test for δ-closeness: Given any function $f : \mathrm{F}^m \to \mathrm{F}$ and $\delta < 0.1$, the procedure tests f for δ-closeness, by examining only $O(1/\delta)$ values of f. If f is a codeword, the procedure accepts with probability 1. If f is not δ-close it rejects with probability $> 1/2$.

(ii) Reconstructing Values of $\widetilde{f}$: Given any δ-close function $f : \mathrm{F}^m \to \mathrm{F}$, and $b \in \mathrm{F}^m$ this procedure outputs $\widetilde{f}(b)$ with probability at least $1 - 2\delta$. It reads the value of f at only

2 points.

Complexity: The procedures examine f in $O(1/\delta)$ random locations which, assuming δ is a constant, requires $O(\log|\mathrm{F}|^m) = O(m)$ random bits. Apart from the time taken to read these values, the procedures perform only $O(1)$ operations in GF(2), which takes only $O(1)$ time.

Next, we describe the procedures for polynomial codes of degree higher than 1. Two differences should be noted. First, the procedures for general degree d require additional information, in the form of a separate table T. (We emphasize that the correctness of the procedure does not depend upon the contents of the table; before using any information from the table, the procedure first checks –probabilistically– that it is correct. Nevertheless, reading information from the table helps efficiency, since checking that it is correct is easier than generating it from scratch.) Second, the total number of entries read by the procedures is some constant independent of the degree d and number of variables m. Even more significantly, in part (ii), the number of entries read is independent of c, the number of points at which the procedure is constructing values of $\widetilde{f}$. The fact that both procedures examine only $O(1)$ entries in the tables will be crucial for our strongest constructions.

Procedure 4.2 (Procedures for the general polynomial code.): Let F be the field GF(q).
(i) Test for δ-closeness: Given $f : \mathrm{F}^m \to \mathrm{F}$, a number d such that $q > 100d^3$, and and table T.
If f is a codeword there exists a table T such that the procedure accepts with probability 1. If f is not δ-close to $\mathrm{F}_d[x_1, \ldots, x_m]$, the procedure rejects with probability at least 1/2 (irrespective of T). The procedure reads $O(1/\delta)$ entries from T and the same number of values of f.

(ii) Reconstructing Values of $\widetilde{f}$: Let c, d be integers satisfying $100cd < q$.
Given: a δ-close function $f : \mathrm{F}^m \to \mathrm{F}$, a sequence of c points $z_1, \ldots, z_c \in \mathrm{F}^m$, and a table T.
The procedure reads 1 values of f and 1 entry from T. If f is a codeword, there exists a table T such that the procedure always outputs the correct values of $\widetilde{f}(z_1), \ldots, \widetilde{f}(z_c)$ (and in particular never outputs REJECT). But otherwise with probability $1 - 2\sqrt{\delta}$ (irrespective of T) the procedure either outputs REJECT, or correct values of $\widetilde{f}(z_1), \ldots, \widetilde{f}(z_c)$.

Complexity: The first procedure runs in time poly$(m + d + \log 1/\delta)$, the second in time poly$(m + d + c)$. Randomness is required only to generate $O(1)$ elements of F^m, so only $O(m \log|F|)$ random bits are needed.

Note: Whenever we use Procedure 4.2, the function f is supposed to represent a sequence of n bits. The field size, the degree and the number of variables have been carefully chosen so that $|\mathrm{F}|^m = \mathrm{poly}(n)$. Thus the procedures require $O(m \log|\mathrm{F}|) = O(\log n)$ random bits. Also, the degree, the number of variables (and c, the number of points in Procedure 4.2-(ii)) is poly$(\log n)$, so both the running time and the size of the table entries are poly$(\log n)$.

4.1.2. An Application: Aggregating Queries

As an immediate application of Procedure 4.2 we prove a general result about how to modify any verifier so that all its queries to the proof get aggregated into $O(1)$ queries. The modification causes a slight increase in the number of random bits and the alphabet size.

Lemma 4.4: *Let L be any language. For every normal form verifier V for L that uses $O(r(n))$ random bits and has decision time poly$(t(n))$ there is a normal form verifier V' for L that is $(r(n)(1+\frac{\log t(n)}{\log r(n)}),\ 1,\ t(n)+r(n))$-restricted.*

Proof: (For starters, we ignore the property of normal form verifiers having to do with checking assignments that are split into many parts.)

The main idea in the construction of the new verifier V' is that the proof is now supposed to be in a different format. It contains a polynomial extension of a string of bits that the old verifier V would have accepted as a proof with probability 1. What enables V' to "bunch up" its queries to a proof in this format is the magical ability of Procedure 4.2-(ii) to extract many values of a provided polynomial by reading only $O(1)$ entries in some accompanying tables.

Now we state the construction more precisely. Let us fix an input x of size n. Let R and t stand for the number of random bits and the decision time of V respectively. Note that $R = O(R(n)), t = \text{poly}(t(n))$. The number of bits of information that V can read and process is upperbounded by the decision time, t. We will assume w.l.o.g. that exactly t bits are read in each run. Since there are only 2^R different possible runs (one for each choice of the random string), and in each run, exactly t bits are read, we may assume that every provided proof-string has size N, where $N \le 2^R t$. Let us call a string in $\{0,1\}^N$ as *perfect* if V accepts it with probability 1.

Assume (by allowing the proof to contain unnecessary bits if necessary) that $N = (l+1)^j$ for some integers j, l. Let F be a field such that $[0,l] \subseteq \mathrm{F}$ (we specify the values of the paramaters later). Recall from Definition 4.3 that every string s in $\{0,1\}^N$ (in other words, s is a proof that the old verifier can check) has a polynomial extension $\widehat{s}$ in $\mathrm{F}_{jl}[x_1,\ldots,x_l]$. The sequence of values that $\widehat{s}$ takes on $[0,j]^l$ is exactly s.

The new verifier V' expects, as a proof that the input is in the language, a function that is a polynomial extension of a perfect string. But given such a function, say $g : \mathrm{F}^l \to \mathrm{F}$, how to check that it represents a perfect string? First, the verifier checks – using Procedure 4.2-(i) – that g is δ-close for some small enough δ. Then it must check whether the sequence of values of $\widetilde{g}$ on $[0,j]^l$ is a perfect string. (Admittedly, the sequence of values of $\widetilde{g}$ is a string of field elements and not of bits, but we can view it as a string of bits by interpreting the zero of the field as the bit 0 and every non-zero as the bit 1.) To do this, V' runs the old verifier V on the sequence, and accepts iff V rejects. Since V queries t bits while checking a proof, the new verifier V' needs to reconstruct the values taken by $\widetilde{g}$ at some t points in $[0,l]^j$. It uses Procedure 4.2-(ii) for this purpose, and therefore needs to read only $O(1)$ entries in some provided table.

Our informal description of V' is complete. A more formal description appears on the next page. Next, we prove that V' is a correct verifier for language L.

If $x \in L$, there is a perfect string. When V' is provided with a polynomial extension of this string, along with all the proper tables required by Procedure 4.2, it accepts with probability 1.

Program of V'.

Given: $f : \mathrm{F}^j \to \mathrm{F}$, tables P and $P_1, \ldots, P_{2^R}$.

(i) Use Procedure 4.2-(i) and table P to check that f is 0.01-close.
    if the procedure fails
            /⋆ f is not a polynomial
        output REJECT and **exit**
(ii) Pick a random $r \in \{0,1\}^R$
    Compute the sequence of t locations in $[1, N]$ that
    V would examine in a proof using r as random string
        /⋆ (As indicated above, these locations may be
        /⋆ viewed as points in $[0, l]^j$.)
    Use Table P_r and Procedure 4.2-(ii) to reconstruct
    the values of $\tilde{f}$ on these t locations.
        /⋆ (As indicated above, these values may be
        /⋆ viewed as bits.)
    If the Procedure rejects,
        REJECT
    else
        simulate V's computation on these t bits
        and ACCEPT iff it accepts
exit

We show that when $x \notin L$ then V' rejects every proof with probability at least $1/2 - 2\sqrt{0.01}$. Let $(f, P, P_1, \ldots, P_{2^R})$ be any proof given to V'. If f is not 0.01-close, part (i) of the program rejects with probability at least $1/2$. So assume w.l.o.g. that f is 0.01-close. Let Γ be the string of bits whose extension is $\tilde{f}$. Since $x \notin L$, the old verifier V must reject Γ with probability at least $1/2$. But the new verifier merely simulates V on Γ, except that the simulation may sometimes be erroneous if Procedure 4.2-(ii) fails to produce the correct bits of Γ. The probability that Procedure 4.2-(ii) outputs an erroneous answer is at most $2\sqrt{0.01}$, since f is 0.01-close. Thus the probability that the new verifier V' rejects while simulating V is at least $1/2 - 2\sqrt{0.01}$.

Hence we have shown the correctness of V as a verifier for language L.

Complexity. The properties of Procedure 4.2 imply that V' reads only $O(1)$ entries from the provided tables. By viewing each table entry as a symbol in some alphabet, we conclude that V' reads only $O(1)$ symbols from the proof. Now we specify the minimum field size that makes everything else work out. Recall that $N = 2^R t$. Assume $l = \text{poly}(\log N)$. Since $l^j \approx N$, this means $j = \Theta(\log N / \log\log N)$. Let field size q be the larger one of $\text{poly}(jl)$ and $\text{poly}(jT)$. This is the minimum we need for Procedure 4.2 to work as claimed.

The decision time of V' is (Decision time of V) + (Time for procedure 4.2), which is $T + \text{poly}(jlT) = \text{poly}(R + T)$. The amount of randomness it uses is

$$R + O(\log |\mathrm{F}|^j) = O(R + j \log T) = O(R + \frac{R \log T}{\log R}).$$

Thus V' respects the claimed bounds.

Checking assignments split into p parts. The modification is the same. That is to say, V' expects each of the p split parts to be represented by its polynomial extension, along with tables similar to the ones above.

□

4.2. A Verifier Using $O(\log n)$ Random Bits

In this section we prove Theorem 3.2, one of the two theorems we wanted to prove in this chapter. One essential ingredient is the following lemma, whose proof appears in Section 4.2.1.

Lemma 4.5: *There exists a normal form verifier for 3SAT that is* $(\log n, \textit{poly}(\log n), \log n)$-*restricted.*

Theorem 3.2 follows easily from this lemma by using the general procedure for aggregating queries into $O(1)$ queries.

Proof: (*Theorem 3.2*) The verifier of Lemma 4.5 fails the requirements of Theorem 3.2 in just one way: it reads $\text{poly}(\log n)$ symbols in the proof instead of $O(1)$. However, Lemma 4.4 allows us to aggregate the queries of this verifier, and replace them with $O(1)$ queries. A quick look at the accompanying blow-up in decision time and amount of randomness shows that it is not too much.

Using Lemma 4.4 on the verifier of Lemma 4.5, we obtain a $(\log n, 1, \log n)$-restricted verifier. □

4.2.1. A Less Efficient Verifier

This section contains a proof of Lemma 4.5. The exposition of the proof uses a mix of ideas from [BFL91, BFLS91, FGL+91].

Let φ be an instance of 3SAT and F a finite field. The verifier will use the fact (see Definition 4.3) that every assignment can be encoded by its polynomial extension.

Definition 4.6: A *satisfying polynomial* for 3SAT instance φ is a polynomial extension of a satisfying assignment for φ. *(Note: see Definition 4.5 for a more detailed definition.)*

Definition 4.7: The *sum* of a function $g : \mathrm{F}^k \to \mathrm{F}$ on a set $S \subseteq \mathrm{F}^k$ is the value $\sum_{x \in S} g(x)$.

The verifier expects the proof to contain a satisfying polynomial, for some specified choice of parameters m, h. Using the procedure described in Section 4.1.1, the verifier checks that the provided function f is δ-close. Next, it has to check that $\widetilde{f}$, the polynomial nearest to f, is a satisfying polynomial. The verifier reduces this question probabilistically (using Lemma 4.6) to whether or not a certain related polynomial P sums to 0 on a certain "nicely behaved" set S. (A set S is "nicely behaved" if it is of the type $[0, l]^i$ for some integers l, i.) The Sum-check procedure (Procedure 4.3) can efficiently verify the sum of a polynomial on a "nicely behaved" set. The verifier uses this procedure to verify that the sum of polyomial P on S is 0; if it is not, the verifier rejects. (While doing the Sum-check, the verifier needs to reconstruct values of $\widetilde{f}$, which is also easy to do using the the procedure in Section 4.1.1.)

Now we set out the parameters used in the rest of this section. Assume m, h are integers such that $h = O(\log n)$, $n = (h+1)^m$ (if n is not of this form, we add unnecessary variables and clauses until it is). Note that this means $m = \theta((\log n)/\log\log n)$. Finally, let $\mathrm{F} = \mathrm{GF}(q)$ be a field of size poly(h). Then a function $f : \mathrm{F}^m \to \mathrm{F}$ is represented by $q^m = h^{O(m)} = \mathrm{poly}(n)$ values.

The following lemma describes an algebraic condition that characterizes a satisfying polynomial.

Lemma 4.6 (Algebraic View of 3SAT): *Given $A \in F_{mh}[x_1, \ldots, x_m]$, there is a polynomial time constructible sequence of poly(n) polynomials $P_1^A, P_2^A, \ldots \in F_{10mh}[x_1, \ldots, x_{4m}]$ such that*

1. *If A is a satisfying polynomial for φ then each of $P_1^A, P_2^A, \ldots$ sums to 0 on $[0, h]^{4m}$. Otherwise at most $1/8$th of them do.*

2. *For each i, evaluating P_i^A at any point requires the values of A at 3 points.*

Proof: In Lemma 4.3 we replaced clauses of φ by cubic equations. A sequence of field elements satisfies these equations iff it represents a satisfying assignment to φ. In this lemma we replace that set of equations by a more compactly stated algebraic condition.

Since $(h+1)^m = n$, the cube $[0,h]^m$ has n points. By definition, polynomial A is a satisfying polynomial iff the sequence of values $(A(v) : v \in [0,h]^m)$ (ordered in some canonical way) represents a satisfying assignment, in other words, satisfy the set of cubic equations in Lemma 4.3.

For $j = 1,2,3$, let $\chi_j(c,v)$ be the function from $[0,h]^m \times [0,h]^m$ to $\{0,1\}$ such that $\chi_j(c,v) = 1$ if v is the $j^{\underline{\text{th}}}$ variable in clause c, and 0 otherwise. Similarly let $s_j(c)$ be a function from $[0,h]^m$ to $\{0,1\}$ such that $s_j(c) = 1$ if the $j^{\underline{\text{th}}}$ variable of clause c is unnegated, and 0 otherwise. The following is a restatement of the set of equations in Lemma 4.3: A is a satisfying polynomial iff for every clause $c \in [0,h]^m$ and every triple of variables $v_1, v_2, v_3 \in [0,h]^m$, we have

$$\prod_{j=1}^{3} \chi_j(c, v_j) \cdot (s_j(c) - A(v_j)) = 0, \tag{4.2}$$

that is to say, iff

$$\prod_{j=1}^{3} \widehat{\chi_j}(c, v_j) \cdot (\widehat{s_j}(c) - A(v_j)) = 0, \tag{4.3}$$

where in the previous condition we have replaced functions χ_j and s_j appearing in condition (4.2) by their polynomial extensions, $\widehat{\chi_j} : \mathrm{F}^{2m} \to \mathrm{F}$ and $\widehat{s_j} : \mathrm{F}^m \to \mathrm{F}$ respectively. Conditions (4.3) and (4.2) are equivalent because by definition, a polynomial extension takes the same values on the underlying cube (which is $[0,h]^m$ for s_j and $[0,h]^{2m}$ for χ_j) as the original function.

Let $g_A : \mathrm{F}^{4m} \to \mathrm{F}$, a polynomial in $\mathrm{F}_{6mh}[x_1, \ldots, x_{4m}]$, be defined as

$$g_A(\overline{z}, \overline{w_1}, \overline{w_2}, \overline{w_3}) = \prod_{j=1}^{3} \widehat{\chi_j}(\overline{z}, \overline{w_j}) \cdot (\widehat{s_j}(\overline{z}) - A(\overline{w_j})) \tag{4.4}$$

where each of $\overline{z}, \overline{w_1}, \overline{w_2}, \overline{w_3}$ is a vector of m variables: $\overline{z} = (x_1, \ldots, x_m)$, $\overline{w_1} = (x_{m+1}, \ldots, x_{2m})$, $\overline{w_2} = (x_{2m+1}, \ldots, x_{3m})$, $\overline{w_3} = (x_{3m+1}, \ldots, x_{4m})$.

Then we may restate condition (4.3) as: A is a satisfying polynomial iff

$$g_A \text{ is 0 at every point of } [0,h]^{4m} \tag{4.5}$$

in other words, iff for every polynomial R_i in the "zero-tester" family we will construct in Lemma 4.7,

$$R_i \cdot g_A \text{ sums to 0 on } [0,h]^{4m}, \tag{4.6}$$

Further, if the condition in (4.5) is false, the statement of Lemma 4.7 implies that the condition (4.6) is false for at least 7/8 of the polynomials in the "zero-tester" family.

Now define the desired family of polynomials $\left\{P_1^A, P_2^A, \ldots, \right\}$ by

$$P_i^A(\overline{z}, \overline{w_1}, \overline{w_2}, \overline{w_3}) = R_i(\overline{z}, \overline{w_1}, \overline{w_2}, \overline{w_3})\; g_A(\overline{z}, \overline{w_1}, \overline{w_2}, \overline{w_3})$$

where R_i is the ith member of the "zero-tester" family. Note that $P_i^A \in F_{10mh}[x_1, \ldots, x_{4m}]$. Further, evaluating P_i^A at any point requires the value of g_A at one point, which (by inspecting (4.4)) requires the value of A at three points.

Thus the claim is proved.

Constructibility. The construction of the polynomial extension in the the proof of Theorem 4.2 is effective. We conclude that the functions $\widehat{\chi_j}, \widehat{s_j}$ can be constructed in poly(n) time.

Thus, assuming Lemma 4.7, Lemma 4.6 has been proved.

□

The following lemma concerns a family of polynomials that is useful for testing whether or not a function is identically zero on the cube $[0, h]^j$ for any integers h, j.

Lemma 4.7: *["Zero-tester" Polynomials, [BFLS91, FGL+91]] There exists a family of $q^{O(m)}$ polynomials $\{R_1, R_2, \ldots\}$ in $F_{4mh}[x_1, \ldots, x_{4m}]$ such that if $f : [0, h]^{4m} \to F$ is any function not identically* 0*, then if R is chosen randomly from this family,*

$$\Pr[\sum_{y \in [0,h]^{4m}} R(y) f(y) = 0] \leq \frac{1}{100}. \tag{4.7}$$

This family is constructible in $q^{O(4m)}$ time.

Proof: In this proof we will use the symbols $0, 1, \ldots, h$ to denote both integers in $\{0, \ldots, h\}$, and field elements. We use boldface to denote the latter use. Thus $\mathbf{0} \in F$ for example.

For now let $t_1, \ldots, t_{4m}$ be formal variables (later we give them values). Consider the following degree h polynomial in $t_1, \ldots, t_{4m}$.

$$\sum_{\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}} \in [\mathbf{0}, \mathbf{h}]} f(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) \prod_{j=1}^{4m} t_j^{i_j}. \tag{4.8}$$

This polynomial is the zero polynomial iff f is identically $\mathbf{0}$ on $[\mathbf{0}, \mathbf{h}]^{4m}$. Further, if it is not the zero polynomial then its roots constitute a fraction no more than $4hm/q$ of all points in F^{4m}. Assume that this fraction is less than $1/100$.

We prove the lemma by constructing a family of q^{4m} polynomials, $\{R_{b_1, \ldots, b_{4m}} : b_1, \ldots, b_{4m} \in F\}$, such that

$$\sum_{(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) \in [\mathbf{0}, \mathbf{h}]} R_{b_1, \ldots, b_{4m}}(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) f(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) = \mathbf{0}$$

iff $(b_1, \ldots, b_{4m})$ is a root of the polynomial in (4.8).

Denote by $I_{t_i}(x_i)$ the univariate degree-h polynomial in x_i whose values at $\mathbf{0}, \mathbf{1}, \ldots, \mathbf{h} \in \mathrm{F}$ are $\mathbf{1}, t_i, \ldots, t_i^h$ respectively (such a polynomial exists, see Fact A.3).

Let g be the following polynomial in variables $x_1, \ldots, x_{4m}, t_1, \ldots, t_{4m}$.

$$g(t_1, \ldots, t_{4m}, x_1, \ldots, x_{4m}) = \prod_{i=1}^{4m} I_{t_i}(x_i).$$

Note that

$$\sum_{\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}} \in [\mathbf{0}, \mathbf{h}]} f(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}})\, g\,(t_1, \ldots, t_m, \mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}})$$
$$= \sum_{\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}} \in [\mathbf{0}, \mathbf{h}]} f\,(\mathbf{i_1}, \mathbf{i_2}, \ldots, \mathbf{i_{4m}}) \prod_{j=1}^{4m} t_j^{i_j}$$

Now define $R_{b_1, \ldots, b_{4m}}$ as the polynomial obtained by substituting $t_1 = b_1, \ldots, t_{4m} = b_{4m}$ in g.

$$R_{b_1, \ldots, b_{4m}}(x_1, \ldots, x_{4m}) = g(b_1, \ldots, b_{4m}, x_1, \ldots, x_{4m}).$$

This family of polynomials clearly satisfies the desired properties. $\square$

Example 4.2: We write a polynomial g for checking the sums of functions on $[\mathbf{0}, \mathbf{h}]^p$ for $\mathbf{h} = \mathbf{1}$ and $p = 2$.

$$g(t_1, t_2, x_1, x_2) = (\mathbf{1} + x_1(t_1 - \mathbf{1}))(\mathbf{1} + x_2(t_2 - \mathbf{1})). \tag{4.9}$$

Hence we have

$$\sum_{x_1, x_2 \in [\mathbf{0}, \mathbf{1}]^2} f(x_1, x_2) g(t_1, t_2, x_1, x_2) = f(\mathbf{0}, \mathbf{0}) + f(\mathbf{1}, \mathbf{0}) t_1 + f(\mathbf{0}, \mathbf{1}) t_2 + f(\mathbf{1}, \mathbf{1}) t_1 t_2. \tag{4.10}$$

Clearly, the polynomial in (4.10) is nonzero if any of its four terms is nonzero.

Now we give a black-box description of Sum-check, a procedure that checks the sums of polynomials on the cube $[0, p]^l$ for some integers p, l. Section 4.4.1 describes the procedure more completely.

Procedure 4.3 (Sum-Check): Let $F = \mathrm{GF}(q)$ and d, l be integers satisfying $2dl < q$. Given: $B \in \mathrm{F}_d[y_1, \ldots, y_l]$, $p \in \mathrm{F}$, a value $c \in \mathrm{F}$, and a table T.
If the sum of B on $[0..p]^l$ is not c, the procedure rejects with probability at least 1/2. But if the sum is c, there is a table T such that the procedure accepts with probability 1.

Complexity. The procedure uses the value of B at one random point in F^l and read another $O(ld \log |\mathrm{F}|)$ bits from the proof. It runs in time $\mathrm{poly}(l + d + \log |\mathrm{F}|)$.

Now we prove Lemma 4.5.

Proof: (*Of Lemma 4.5*) (For now we ignore the aspect of normal form verifiers having to do with checking split assignments.)

Our verifier expects the proof to contain a function $f : \mathrm{F}^m \to \mathrm{F}$, and a set of tables described below. (*In a good proof, f is a satisfying polynomial.*)

One of these tables allows the verifier to check that f is 0.01-close (using Procedure 4.2-(i)).

Another set of tables allows the verifier to perform a Sum-check on each polynomial in the family $\left\{P_1^{\tilde{f}}, P_2^{\tilde{f}}, \ldots,\right\}$ constructed in Lemma 4.6 concerning the algebraic view of 3SAT.

Another set of tables allows the verifier to reconstruct the value of $\tilde{f}$ at any three points.

The verifier works as follows. It checks that $\tilde{f}$ is 0.01-close. Then, to checks that $\tilde{f}$ is a satisfying polynomial, it uses the algebraic view of 3SAT . It uses $O(\log n)$ random bits to select a polynomial P uniformly at random from the family of Lemma 4.6, and uses the Sum-check (Procedure 4.3) to check that P sums to 0 on $[0,h]^{4m}$. (We have described the verification as a sequence of steps, but actually no step requires results from the previous steps, so they can all be done in parallel.)

The Sum-check requires the value of the selected polynomial P at one point, which by the statement of Lemma 4.6 requires values of $\tilde{f}$ at 3 points. The verifier reconstructs these three values using Procedure 4.2-(ii), and the appropriate table in the proof.

Correctness: Suppose φ is satisfiable. The verifier clearly accepts with probability 1 any proof containing the polynomial extension of a satisfying assignment, as well as proper tables required by the various procedures.

Now suppose φ is not satisfiable. If f is not 0.01-close, the verifier rejects with probability at least 1/2. So assume w.l.o.g. that f is 0.01-close. The verifier can accept only if one of the three events happens. (i) The selected polynomial P sums to 0 on $[0,h]^{4m}$. By Lemma 4.6 this event can happen with probability at most 1/8. (ii) P does not sum to 0, but the Sum-check fails to detect this. The probability of this event is upperbounded by the error probability of the Sum-check, which is $O(mh/q) = o(1)$. (iii) Procedure 4.2-(ii) produces erroneous values of $\tilde{f}$. The probability of this event is upperbounded by $2\sqrt{0.01} \leq 0.7$.

To sum up, if φ is not satisfiable, the probability that the verifier accepts is at most $1/8 + 0.7 + o(1)$, which is less than 1. By running it $O(1)$ times, the probability can be reduced below 1/2.

Complexity: By inspecting the complexity of the Sum-check, the low-degree test, and the test for δ-closeness, we see that the verifier needs only $\log |\mathrm{F}|^{4m} = O(\log n)$ random bits for its operation.

To achieve a low decision time the verifier has to do things in a certain order. In its first stage it reads the input, selects the above-mentioned polynomial P and constructs P as outlined in the proof of Lemma 4.6. All this takes poly(n) time, and does not involve reading the proof. The rest of the verification requires reading the proof, and consists of the following procedures: the test for 0.01-closeness, the Sum-check, and the reconstruction of three values of $\widetilde{f}$. All these procedures run in time polynomial in the degree h, the number of variables $O(m)$ and the log of the field size, $\log |\mathrm{F}|$. We chose these parameters to be poly($\log n$). Hence the decision times is poly($\log n$), and so is the alphabet size and the number of queries.

To finish our claim that the verifier is in normal form, we have to show that it can check split assignments. We do this next. □

4.2.2. Checking Split Assignments

We show that the verifier of Lemma 4.5 can check assignments that are split into k parts for any positive integer k.

Recall (from Definition 3.2) that in this setting the verifier defines an encoding method σ, and expects the proof to be of the form $\sigma(S_1)\circ\cdots\circ\sigma(S_k)\circ\pi$, where π is some information that allows an efficient check that $S_1\circ\cdots\circ S_k$ is a satisfying assignment ($\circ$ = concatenation of strings).

In this case we assume the 3SAT instance φ has nk variables, split into k equal blocks: $(y_1,\ldots,y_n)$, $\ldots$, $(y_{n(k-1)+1},\ldots,y_{nk})$.

Let $n=(h+1)^m$, with m,h the same as in the proof of Lemma 4.5. The verifier uses the encoding σ that maps strings in $\{0,1\}^n$ to their degree-mh polynomial extensions. In other words the proof is required to contain k functions $f_1,\ldots,f_k:\mathrm{F}^m\to\mathrm{F}$. Part π has to contain a set of tables. (*In a good proof, $f_1,\ldots,f_k$ are m-variate extensions of assignments to the k blocks, such that the overall assignment is a satisfying assignment.*)

While checking such a proof, the verifier follows the program of Lemma 4.5 quite closely. It first checks that each $f_1,\ldots,f_k$ is 0.1-close. Then, using a modification of the algebraic view of 3SAT (Lemma 4.6), it reduces the question of whether or not $\widetilde{f_1}\ldots,\widetilde{f_k}$ together represent a satisfying assignment to a single Sum-check. The modification is the following.

Corollary 4.8: *Given $A_1,\ldots,A_k\in F_{mh}[x_1,\ldots,x_m]$, there is a sequence of polynomials $P_1^{A_1,\ldots,A_k}$, $P_2^{A_1,\ldots,A_k}$, $\ldots,\in F_{10mh}[x_1,\ldots,x_{4m}]$ such that*

1. *If $A_1,\ldots,A_k$ together represent a satisfying assignment then all of $P_1^{A_1,\ldots,A_k}$, $P_2^{A_1,\ldots,A_k}$, $\ldots$ sum to 0 on $[0,h]^{4m}$ and otherwise at most 1/8 of them do.*

2. *the value of $P_i^{A_1,\ldots,A_k}$ at any point can be constructed from the values of $A_1,A_2,\ldots,A_k$ at 3 points each.*

Proof: For $j = 1, 2, 3$ and $i = 1, 2, \ldots, k$ let p_{ij} be a function from $[0, h]^m \times [0, h]^m$ to $\{0, 1\}$ such that $p_{ij}(c, v)$ is 1 iff $y_{(i-1)n+v}$ is the j'th variable in clause c. (Note: p_{ij} is some kind of "multiplexor" function.)

Use the same arithmetization as in lemma 4.6, except replace $A(v_j)$ by $\sum_{i=1}^{k} \widehat{p_{ij}}(v_j) A_i(v_j)$.
□

4.3. A Verifier using $O(1)$ query bits

Now we prove Theorem 3.4 about the existence of a $(n^3, 1, 1)$-restricted normal form verifier for 3SAT. In this section G denotes the field GF(2) and $+$, $\cdot$ denote operations in G. (We will often write $x \cdot y$ as xy where this causes no confusion.) Let φ denote a 3CNF formula that is the verifier's input, and let n be the number of variables and clauses in it.

A function $f : \mathrm{G}^k \to \mathrm{G}$ is called *linear* if for some $a_1, \ldots, a_k \in \mathrm{G}$ it can be described as $f(x_1, \ldots, x_k) = \sum_{i=1}^{k} a_i x_i$. Since each coeficient a_i can take only two values, the set of n-variate linear functions is in one-to-one correspondence with the set of possible assignments to φ.

Definition 4.8: A linear function $\sum_{i=1}^{n} a_i \cdot x_i$ is called a *satisfying linear function for* φ if its coefficients $a_1, \ldots, a_n$ constitute a satisfying assignment to φ (where we view $1 \in G$ and $0 \in G$ as the boolean values T, F respectively).

The verifier expects the proof to contain a satisfying linear function, plus some other information. As in Section 4.1, we assume that functions are represented by a table of their values at all points in the field. A satisfying linear function is represented by $|\mathrm{G}|^n = 2^n$ values[2].

In its basic outline, this verifier resembles the one in Lemma 4.5. First it checks that the provided function, f, is 0.01-close to the set of linear functions. Then it writes down a sequence of algebraic conditions which characterize satisfying linear functions, and checks that $\tilde{f}$, the linear function closest to f, meets these conditions. Instead of the Sum-check, it uses Fact 4.10.

The following lemma describes the set of algebraic conditions.

Lemma 4.9: *Let φ be a 3SAT instance with n variables and n clauses and let $X_1, \ldots, X_n$ be variables taking values over $GF(2)$. There is a poly(n) time construction of n cubic equations $\{p_i(X_{i_1}, X_{i_2}, X_{i_3}) = 0 : 1 \le i \le n\}$, such that any linear function $\sum_{i=0}^{n} a_i x_i$ is a satisfying linear function for φ iff*

$$p_i(a_{i_1}, a_{i_2}, a_{i_3}) = 0 \quad \forall i \in [1, n] \tag{4.11}$$

[2]There is no problem with the proof being of size 2^n. Recall that the verifier has random access to the proof string, so it requires only $O(\log 2^n) = O(n)$ time to access any bit in this proof.

Proof: Identical to that of Lemma 4.3. $\Box$

The following fact about a non-zero vector of bits forms the basis of the verifier's probabilistic tests.

Fact 4.10: *Let $c_1, \ldots, c_k$ be elements of G that are not all zero. Then for $r_1, \ldots, r_k$ picked randomly and independently from G,*

$$\Pr_{r_1,\ldots,r_k}[\sum_i c_i \cdot r_i \neq 0] = \frac{1}{2}.$$

Proof: Assume w.l.o.g. that $c_k \neq 0$. After $r_1, \ldots, r_{k-1}$ have been picked, the sum $\sum_i c_i \cdot r_i$ is still equally likely to be 0 or 1. $\Box$

We describe an application of the above fact. We need the following definition.

Definition 4.9: Let b, c be the following linear functions in k and l variables respectively, $\sum_{i=1}^{k} b_i x_i$ and $\sum_{j=1}^{l} c_i x_i$. The *tensor product* of b and c, denoted $b \otimes c$, is the following linear function in kl variables[3]: $\sum_{i=1}^{k}\sum_{j=1}^{l} b_i c_j z_{ij}$.

The following lemma implicitly gives a procedure for testing, given linear functions b, c, d, whether or not d is the tensor product of b and c. The probabilistic test appearing in the statement of the lemma requires one value each of all three functions.

Lemma 4.11 (Testing for Tensor Product): *Let b, c, d be the following linear functions in k, l and kl variables respectively: $\sum_{i=1}^{k} b_i x_i$, $\sum_{j=1}^{l} c_i x_i$ and $\sum_{i=1}^{k}\sum_{j=1}^{l} d_{ij} z_{ij}$ If $d \neq b \otimes c$ then for $u_1, \ldots, u_k$, and $v_1, \ldots, v_l$ chosen randomly from G*

$$\Pr_{u_1,\ldots,v_1,\ldots,}[\sum_{i,j} d_{ij} \cdot u_i \cdot v_j \neq (\sum_i b_i \cdot u_i)(\sum_j c_j \cdot v_j)] \geq 1/4$$

(where the indices i, j take values in $[1, k]$ and $[1, l]$ respectively).

Proof: Consider the two $k \times l$ matrices M, N defined as $M_{(i,j)} = (d_{ij})$ and $N_{(i,j)} = (b_i \cdot c_j)$. If $d \neq b \otimes c$ then $M \neq N$. Let a vector $\widehat{u} = (u_1, \ldots, u_k)$ be picked randomly from G^k. Fact 4.10 implies that with probability at least 1/2,

$$\widehat{u}M \neq \widehat{u}N$$

(where $\widehat{u}M$ stands for the product of vector $\widehat{u}$ with the matrix M as defined normally).

[3]In normal mathematical usage, d would be called the linear function whose sequence of coefficients is the tensor product of the sequence of coefficients of b with the sequence of coefficients of c.

Now let vector $\widehat{v} = (v_1, \ldots, v_l)$ be also picked randomly from G^l. Fact 4.10 implies that with probability at least 1/4,

$$\widehat{u}^T M \widehat{v} \neq \widehat{u}^T N \widehat{v}.$$

Hence the lemma is proved. $\Box$

Now we prove Theorem 3.4.

Proof: *(Theorem 3.4)* The verifier expects the proof to contain tables of three functions, $f : G^n \to G$, $g : G^{n^2} \to G$, and $h : G^{n^3} \to G$. (*In a good proof, f is a satisfying linear function and $g = f \otimes f$, $h = f \otimes g$.*) For ease of exposition, we describe the verification as consisting of a sequence of three steps. However, the verifier can actually do all three in parallel, since no step uses the results of the previous step.

Step 1. The verifier runs Procedure 4.1-(i) to check f, g, h for 0.01-closeness, and rejects outright if that procedure rejects any of f, g, h.

Procedure 4.1-(i) reads only $O(1)$ values each of f, g, h, and rejects with high probability if either of the three is not 0.01-close. Assume for argument's sake that the probability that it rejects is not high. Then all three functions are 0.01-close. Let $\widetilde{f}$, $\widetilde{g}$, $\widetilde{h}$ be the linear functions closest to f, g, h respectively.

Step 2. The verifier runs a test that rejects with high probability if either $\widetilde{g} \neq \widetilde{f} \otimes \widetilde{f}$, or $\widetilde{h} \neq \widetilde{f} \otimes \widetilde{g}$.

The statement of Lemma 4.11 implicitly shows how to do Step 2. For instance, to check $\widetilde{g} = \widetilde{f} \otimes \widetilde{f}$, the verifier can repeat the following test $O(1)$ times: Pick random $(u_1, \ldots, u_n)$ and $(v_1, \ldots, v_n)$, and check that

$$\widetilde{g}((u_i v_j)) = \widetilde{f}(u_1, \ldots, u_n)\widetilde{f}(v_1, \ldots, v_n),$$

where $(u_i v_j)$ is used as shorthand for a vector of length n^2 whose (i, j) coordinate for $i, j \in [1, n]$ is $u_i v_j$.

For this test the verifier needs to reconstruct values of $\widetilde{f}$ at 2 points and that of $\widetilde{g}$ at 1 point. It uses Procedure 4.1-(ii), to do this, while reading the values of f, g, h at only $O(1)$ points in the process.

Assume for argument's sake that Step 2 does not reject with high probability. Then $\widetilde{g} = \widetilde{f} \otimes \widetilde{f}$ and $\widetilde{h} = \widetilde{f} \otimes \widetilde{g}$. In other words, if the $\widetilde{f} = \sum_i a_i x_i$, then

$$\widetilde{g}((y_{ij})) = \sum_{i,j} a_i a_j y_{ij} \quad \text{and}$$

$$\widetilde{h}((z_{ijk})) = \sum_{i,j,k} a_i a_j a_k z_{ijk},$$

where $i, j, k \in [1, n]$.

Step 3. The verifier runs a test that rejects with high probability if $\widetilde{f}$ is not a satisfying linear function, that is, its coefficients $a_1, \ldots, a_n$ do not satisfy condition (4.11) in Lemma 4.9.

The verifier picks a random vector $(r_1, \ldots, r_n) \in \mathrm{G}^n$, and checks that

$$\sum_{i=1}^{n} r_i \cdot p_i(a_{i_1}, a_{i_2}, a_{i_3}) = 0.$$

Suppose $a_1, \ldots, a_n$ are such that the n-bit vector $(p_i(a_{i_1}, a_{i_2}, a_{i_3}) : \; i \in \{1, \ldots, n\})$ is not the zero vector. Then Fact 4.10 implies that the test rejects with probability 1/2.

But how can the verifier compute the sum $\sum_{i=1}^{n} r_i \cdot p_i(a_{i_1}, a_{i_2}, a_{i_3})$? Note that the sum is a linear combination of the p_i's, which since the p_i's are cubic, can be expressed as the sum of one value each of the functions $\sum_i a_i x_i$, $\sum_{i,j} a_i a_j y_{ij}$, and $\sum_{i,j,k} a_i a_j a_k z_{ijk}$. In other words, the verifier only needs to reconstruct one value each of $\widetilde{f}$, $\widetilde{g}$, $\widetilde{h}$ respectively. This is easy.

This finishes the description of the verifier.

Complexity. Steps 1, 2 and 3 require reading only $O(1)$ bits from the proof. They require $O(\log |\mathrm{G}|^{n^3})$ random bits, in other words, $O(n^3)$ random bits.

Correctness: Suppose formula φ is satisfiable. It is clear that Steps 1, 2 and 3 never reject a proof containing a satisfying linear extension and its tensor products.

Now suppose φ is not satisfiable, and so there is no satisfying linear function either. One of the following must be true: one of f, g, h is not 0.01-close, or the tensor-product property is violated, or the condition in (4.1) is violated. In each case, one of Steps *1, 2* or *3* rejects with probability at least 1/2.

We still have to show how the verifier can check assignments split into many parts, as required by Definition 3.2. We show this next. □

4.3.1. Checking Split Assignments

We sketch how the verifier of Theorem 3.4 can also check assignments split into many parts.

Recall (from Definition 3.2) that in this setting the verifier defines an encoding method σ, and expects a proof to be of the form $\sigma(S_1) \circ \cdots \circ \sigma(S_k) \circ \pi$, where π is some information that allows an efficient check that $S_1 \circ \cdots \circ S_k$ is a satisfying assignment ($\circ$ = concatenation).

In this case we assume the 3SAT instance φ has nk variables split into k equal-sized blocks, $(y_1, \ldots, y_n)$, $\ldots$, $(y_{n(k-1)+1}, \ldots, y_{nk})$.

The verifier uses the encoding σ that maps a string $(a_1,\ldots,a_n) \in \{0,1\}^n$ to the linear function $\sum_{i=1}^n a_i x_i$. In other words the proof is required to contain k functions $f_1,\ldots,f_k : \mathrm{F}^m \to \mathrm{F}$. Part π contains a set of tables. (*In a good proof, f_i is a linear function representing an assignment to the variables in the ith block, such that the overall assignment represented by $f_1,\ldots,f_k$ is a satisfying assignment. Part π contains $f \otimes f$ and $f \otimes f \otimes f$, where f is the linear function defined below.*)

The verifier uses Procedure 4.1 to check that $f_1,\ldots,f_k$ are 0.01-close. Suppose the procedure does not reject. Then the verifier defines a function $f : \mathrm{G}^{nk} \to \mathrm{G}$ as

$$f(x_1,\ldots,x_{nk}) = \sum_{i=1}^{k} \widetilde{f}_i(x_{n(i-1)+1},\ldots,x_{ni}).$$

Clearly, f is a linear function and its nk coefficients form a sequence that is the concatenation of the sequences formed by the coefficients of $\widetilde{f_1},\ldots,$ and $\widetilde{f_k}$. Now the verifier uses f in Steps 2 and 3 exactly as before. Clearly, if f is not a satisfying linear function, the verifier rejects with high probability.

From the definition of f it should be clear that only $O(1)$ values each of $f_1, f_2, \ldots, f_k$ need to be read.

4.4. The Algebraic Procedures

Now we describe in some detail the Procedures 4.1, 4.2 and 4.3. Throughout this section F denotes a field. A procedure may work only for certain field sizes, in which case we will specify these sizes explicitly.

4.4.1. Sum-Check

We describe Procedure 4.3, the Sum-Check. The inputs to the procedure consist of a degree-d polynomial B in l variables, a set $H \subseteq \mathrm{F}$, and a value $c \in \mathrm{F}$. The procedure has to verify that the sum of the values of B on the subset H^l of F^l is c. It will need, in addition to the table of values of B and the integers l and d, an extra table. We describe first what the table must contain.

For $a \in \mathrm{F}$ let us denote by $B(a, y_2, \ldots, y_l)$ the polynomial obtained from B by fixing the first variable to a.

When we fix all variables of B but one, we get a univariate polynomial of degree at most d in the unfixed variable. It follows that the sum

$$\sum_{y_{i+1},..,y_l \in H} B(a_1,..,a_{i-1},\ y_i,\ y_{i+1},..,y_l) \tag{4.12}$$

for i s.t. $1 \le i < l$, and $a_1, \ldots, a_{i-1} \in \mathrm{F}$, is a degree-$d$ univariate polynomial in the variable y_i. We denote this sum by $B_{a_1,..,a_{i-1}}(y_i)$. (For $i = 1$ the notation $B_{a_0}(y_1)$ does not make sense, so we use the notation $B_\epsilon(y_1)$ instead.)

Example 4.3: The univariate polynomial $B_\epsilon(y_1)$ is represented by $d+1$ coefficients. When we substitute $y_1 = a$ in this polynomial we get the value $B_\epsilon(a)$, which, by definition, is the sum of B on the following sub-cube:

$$\{(x_1, \ldots, x_l) : x_1 = a, \text{ and } x_2, \ldots, x_l \in H\}.$$

(Alternatively, we can view the value $B_\epsilon(a)$ as the sum of $B(a, y_2, \ldots, y_l)$ on H^{l-1}.)

Thus $B_\epsilon(y_1)$ is a representation of $q = |\mathrm{F}|$ sums using $d+1$ coefficients. Suppose $f(y_1)$ is another degree-d univariate polynomial different from $B_\epsilon(y_1)$. Then the two polynomials agree at no more than d points. Hence for $q - d$ values of a, the value $f(a)$ is *not* the sum of $B(a, y_2, \ldots, y_l)$ on H^{l-1}. This observation is useful in designing the Sum-check.

Definition 4.10: A *table of partial sums* is any table containing for every i, $1 \le i \le l$, and every $a_1, ..., a_{i-1} \in \mathrm{F}$, a univariate polynomial $g_{a_1,..a_{i-1}}(y_i)$ of degree d. The entire table is denoted by g.

Now we describe Procedure 4.3. It expects the table T to be a table of partial sums. (*In a good proof, the table contains the set of polynomials defined in (4.12).*)

Sum-Check

Inputs: $B \in \mathrm{F}_d[x_1, \ldots, x_l]$, $c \in \mathrm{F}$.
To Verify: Sum of B on H^l is c.
Given: Table of partial sums, g.

```
current-value = c
Pick random a_1, ..., a_l ∈ F
For i = 1 to l do
    if current-value ≠ Σ_{y_i ∈ H} g_{a_1,..,a_{i-1}}(y_i)
        output REJECT; exit
    else
        current-value = g_{a_1,..,a_{i-1}}(a_i).
If g(a_1, ..., a_{l-1})(a_l) ≠ B(a_1, ..., a_l)
    output REJECT
else
    output ACCEPT
```

Complexity: The procedure needs $l \log q$ random bits to generate elements $a_1, \ldots, a_l$ randomly from F. It needs the value of B at one point, namely, $(a_1, \ldots, a_l)$. In total, it reads l entries from the table of partial sums, where each entry is a string of size at most $(d+1)\log q$. It performs $O(ldh)$ field operations, where $h = |H|$. Therefore the running time is $\mathrm{poly}(ldh \log q)$.

Correctness: Suppose B sums to c on H^l. The procedure clearly accepts with probability 1 the table of partial sums containing the univariate polynomials B_ϵ, $B_{a_1}(y_2)$, etc. defined in (4.12).

Suppose B does not sum to c. The next lemma shows that the procedure rejects with high probability.

Lemma 4.12: $\forall B \in F_d[x_1, \ldots, x_l], c \in F$, *if* B *does not sum to* c *on* H^l *then*

$$\Pr[\textit{the Sum-check outputs } REJECT] \geq 1 - \frac{dl}{q}$$

regardless of what the table of partial sums contains.

Proof: The proof is by induction on the number of variables, l. Such an induction works because the Sum-check is essentially a recursive procedure: it randomly reduces the problem of checking the sum of a polynomial in l variables to checking the sum of a polynomial in $l-1$ variables.

To see this, view the table of partial sums as a tree of branching factor q (see Figure 4.2). The polynomial $g_\epsilon(y_1)$ is stored at the root of the tree, and the set of polynomials $\{g_{a_1}(y_2) : a_1 \in \mathrm{F}\}$ are stored on the children of the root, and so on.

The first step in the Sum-check verifies that that the sum of the values taken by g_ϵ on the set H is c. Suppose the given multivariate polynomial B does not sum to c on H^l. Then the sum of the values taken by B_ϵ on H is not c, and the first step can succeed only if $g_\epsilon \neq B_\epsilon$. But if $g_\epsilon \neq B_\epsilon$ then, as observed in Example 4.3, $g_\epsilon(a) \neq B_\epsilon(a)$ for $q-d$ vals of a in F. That is to say, for $q-d$ values of a, the value $g_\epsilon(a)$ is *not* the sum of $B(a, y_2, \ldots, y_l)$ on H^{l-1}. Since $d \ll q$ it suffices to picks a value for y_1 randomly out of F, say a_1, and check (recursively) that $B(a_1, y_2, \ldots, y_l)$ sums to $g_\epsilon(a_1)$ on H^{l-1}. (Note: While checking the sum of $B(a_1, y_2, \ldots, y_l)$ on H^{l-1}, the recursive call must use as the table of partial sums the sequence of polynomials stored in the a_1th sub-tree of the root.) This is exactly what the remaining steps of the Sum-check do. In this sense the Sum-check is a recursive procedure.

Now we do the inductive proof.

Base case: $l = 1$. This is easy, since $B(y_1)$ is a univariate polynomial, and $B_\epsilon = B$. The table contains only one polynomial g_ϵ. If $g_\epsilon = B$, then g_ϵ doesn't sum to c either and is rejected with probability 1. If $g_\epsilon \neq B$, then the two disagree in at least $q-d$ points. Therefore $\Pr_{a_1}[g_\epsilon(a_1) \neq B(a_1)] \geq 1 - d/q$.

Inductive Step: Suppose the assumption is true for all polynomials in $l-1$ variables. Now there are two cases.

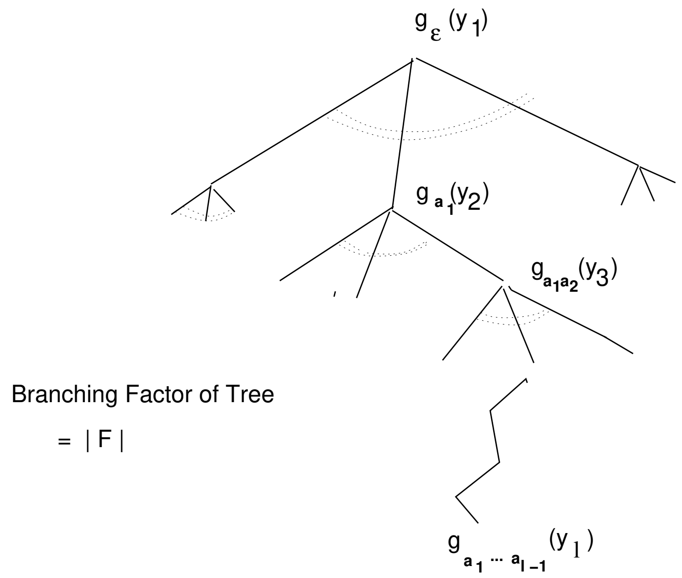

Figure 4.2: A table of partial sums may be conceptualized as a tree of branching factor q. The Sum-check follows a random path down this tree.

Case (i): $g_\epsilon = B_\epsilon$. In this case,

$$\sum_{y_1 \in H} g_\epsilon(y_1) = \sum_{y_1 \in H} B_\epsilon(y_1) \neq c,$$

so the procedure will REJECT rightaway (i.e., with probability 1). So the inductive step is complete.

Case (ii): $g_\epsilon \neq B_\epsilon$. In this case, as observed in Example 4.3, for $q - d$ vals of a,

$$g_\epsilon(a) \neq B_\epsilon(a). \tag{4.13}$$

Let $a_1 \in \mathrm{F}$ be such that $g_\epsilon(a_1)$ is not the sum of $B(a_1, y_2, \ldots, y_l)$ on H^{l-1}. By the inductive assumption, no table of partial sums can convince the Sum-check with probability more than $d(l-1)/q$ that $g_\epsilon(a_1)$ is the sum of $B(a_1, y_2, \ldots, y_l)$ on H^{l-1}. In particular, the table of partial sums stored in the subtree rooted at the a_1th child of the root cannot make the Sum-check accept with probability more than $d(l-1)/q$. Since this is true for $q - d$ values of a_1, the overall probability of rejection is at least $(\frac{q-d}{q})(1 - d(l-1)/q) \geq 1 - dl/q$.

In either case, the inductive step is complete.

□

4.4.2. Procedures for the Linear Function Code

In this section, F denotes the field GF(2). Our first procedure checks whether or not a given function $f : \mathrm{F}^m \to \mathrm{F}$ is δ-close to a linear function. It uses the following obvious property of linear functions. A function $h : \mathrm{F}^m \to \mathrm{F}$ is linear iff for every pair of points $(y_1, \ldots, y_m)$ and $(z_1, \ldots, z_m)$ in F^m it satisfies

$$h(y_1 + z_1, \ldots, y_m + z_m) = h(y_1, \ldots, y_m) + h(z_1, \ldots, z_m). \tag{4.14}$$

(The only if part of the statement is easy; the if part follows from Fact A.6 in the appendix.)

The procedure uses a stronger version of the above statement: if h satisfies the property in 4.14 for "most" pairs of m-tuples, then h is δ-close for some small δ.

Test for δ-closeness; Procedure 4.1-(i).

Given: $f : \mathrm{F}^m \to \mathrm{F}$ where $\mathrm{F} = \mathrm{GF}(2)$.

```
repeat 6/δ times:
    Pick points y, z randomly from F^m.
    if f(y) + f(z) ≠ f(y + z)
            /* Note: + on the left is addition mod 2 and
            /* that on the right is componentwise addition mod 2.
        exit and REJECT
exit and ACCEPT.
```

Complexity: The test requires $12m/\delta$ random bits, and reads $18/\delta$ values of f.

Correctness: Note that if $f \in \mathrm{F}_1[x_1, \ldots, x_m]$ then the test accepts with probability 1. According to the contrapositive to the next lemma, if f is not 3δ-close, then the basic step in the test fails with probability at least δ. Hence, after repeating the basic step $6/\delta$ times, the test rejects with probability close to $1 - 1/e^2$.

Theorem 4.13 ([BLR90]): *Let $F = GF(2)$ and f be a function from F^m to F such that when we pick y, z randomly from F^m,*

$$\Pr[f(y) + f(z) = f(y + z)] \geq 1 - \delta,$$

where $\delta < 1/6$. Then f is 3δ-close to some linear function.

Proof: The proof consists in three claims.

Claim 1: For every point $b \in \mathrm{F}^m$ there is a value $g(b) \in \{0, 1\}$ such that

$$\Pr_{w \in \mathrm{F}^m}[f(w + b) - f(w) = g(b)] \geq 1 - 2\delta.$$

Proof: Let $b \in \mathrm{F}^m$. Denote by p the probability $\Pr_w[f(w+b)-f(w)=1]$, where w is picked uniformly at random in F^m. Define random variables v_1, v_2 (taking values in F) as follows. Pick points $y, z \in \mathrm{F}^m$ randomly and independently from F^m, and let $v_1 = f(y+b) - f(y)$, and $v_2 = f(z+b) - f(z)$. Clearly, v_1, v_2 are independent random variables that take value 1 with probability p and 0 with probability $(1-p)$. The probability of the event "$v_1 = v_2$" is exactly $p^2 + (1-p)^2$. We show that actually this event happens with probability at least $1 - 2\delta$, whence it follows that either $p > 1 - 2\delta$ or $p < 2\delta$. If $p > 1 - 2\delta$, setting $g(b)$ to 1 fulfills the requirements of the lemma; in the other case, setting $g(b)$ to 0 does.

Note that $+$ and $-$ are the same over GF(2), so

$$\begin{aligned} v_1 - v_2 &= f(y+b) - f(y) - (f(z+b) - f(z)) \\ &= (f(z+y+b) - f(y+b)) + (f(z+y+b) - f(z+b)) - f(y) - f(z) \end{aligned}$$

Further, $y+b$ and $z+b$ are independently chosen random points. Hence the probability that each of the following two events happens is at least $1-\delta$: "$f(z+y+b) - f(y+b) = f(z)$", and "$f(z+y+b) - f(z+b) = f(y)$." So the probability that they both happen is at least $1 - 2\delta$, that is,

$$\Pr[(f(z+y+b) - f(y+b)) + (f(z+y+b) - f(z+b)) - f(y) - f(z) = 0] \geq 1 - 2\delta.$$

Thus $\Pr[v_1 = v_2] \geq 1 - 2\delta$, which finishes the proof of Claim 1.

Claim 2: The function g constructed in Claim 1 agrees with f in at least $1 - 3\delta$ fraction of b in F^m.

Proof: Let ρ be the fraction of points $b \in \mathrm{F}^m$ such that $f(b) = g(b)$.

Pick y, z randomly from F^m, and denote by A the event "$f(y+z) = g(y+z)$," and by B the event "$f(y) + f(y) = f(y+z)$." Note that A and B need not be independent. However, the hypothesis of the theorem implies that $\Pr[B] \geq 1 - \delta$. Further our assumption was that $\Pr[A] = \rho$. Now note that

$$\begin{aligned} \Pr[B] &= \Pr[B \wedge A] + \Pr[B \wedge \overline{A}] \\ &\leq \Pr[A] + \Pr[B \mid \overline{A}] \\ &\leq \rho + 2\delta \end{aligned}$$

where the last line uses the following implication of Claim 1:

$$\Pr[\text{“}f(y) + f(z) = f(y+z)\text{”} \mid \text{“}f(y+z) \neq g(y+z)\text{”}] \leq 2\delta.$$

But as we observed, $\Pr[B] \geq 1 - \delta$. Hence $\rho \geq 1 - 3\delta$. This finishes the proof of Claim 2.

Claim 3: Function g is linear, that is

$$\forall a, b \in \mathrm{F}^m, \quad g(a+b) = g(b) + g(a).$$

Proof: Fix arbitrary points $a, b \in \mathrm{F}^m$. To prove $g(a+b) = g(a)+g(b)$, it suffices to prove the existence of points $y, z \in \mathrm{F}^m$ such that each of the following is true: (i) $f(b+a+y+z)-f(y+z) = g(a+b)$ (ii) $f(b+a+y+z)-f(a+y+z) = g(b)$ and (iii) $f(a+y+z)-f(y+z) = f(a)$.

For, if (i), (ii) and (iii) are true for any $y, z \in \mathrm{F}^m$ then

$$\begin{aligned} g(b+a) &= f(b+a+y+z) - f(y+z) \\ &= f(b+a+y+z) - f(a+y+z) + f(a+y+z) - f(y+z) \\ &= g(b) + g(a) \end{aligned}$$

We prove the existence of the desired y, z in a probabilistic fashion. Choose y, z independently at random from F^m. The probability that any of (i), (ii), and (iii) is true is (by Claim 1) at least $1 - 2\delta$, and so the probability that all three are true is at least $1 - 6\delta$. Since $6\delta < 1$, the probability is strictly more than 0 that we obtain a pair y, z satisfying all the conditions of the claim. It follows that the desired pair y, z exists. This proves Claim 3.

Finally, note that Claims 2 and 3 imply (together with the fact in Equation 4.14) that f is $(1 - 3\delta)$-close.

□

Now we describe the other procedure connected with the linear function code.

Producing a value of $\widetilde{f}$; Procedure 4.1-(ii).

Given: $f : \mathrm{F}^m \to \mathrm{F}$ that is δ-close; $\mathrm{F} = \mathrm{GF}(2)$.
Point $b \in \mathrm{F}^m$.

Pick random point y in F^m.
output $f(y+b) - f(y)$.

Complexity: The procedure uses $2m$ random bits and reads 2 values of f.

Correctness: If f is a linear function, then $f = \widetilde{f}$, and $\Pr_y[f(y+b) - f(y) = \widetilde{f}(b)] = 1$.

Now suppose f is just δ-close to some linear function. The following lemma shows that the procedure works correctly.

Lemma 4.14: $\Pr_y[f(y+b) - f(y) = \widetilde{f}(b)] \geq 1 - 2\delta$.

Proof: Both y and $y+b$ are uniformly distributed in F^m (although they are not independent), hence

$$\Pr[f(y) = \widetilde{f}(y)] \geq 1 - \delta \quad \text{and} \quad \Pr[f(y+b) = \widetilde{f}(y+b)] \geq 1 - \delta.$$

Since $\widetilde{f}(b) = \widetilde{f}(b+y) - \widetilde{f}(y)$, we conclude that $\Pr[f(b+y) - f(y) = \widetilde{f}(b)] \geq 1 - 2\delta$. $\Box$

4.4.3. Procedures for General Polynomial Code

Both procedures are randomized; they use randomness only to pick $O(1)$ points uniformly at random from F^m. This requires $O(m \log |\mathrm{F}|)$ random bits. Each procedure also requires that some table be provided in addition to f. The entries in the table have size $\mathrm{poly}(d, m, \log q)$, and the procedure reads $O(1)$ entries from the table.

Procedure 4.2-(i), the Low-degree Test, verifies that a given function $f : \mathrm{F}^m \to \mathrm{F}$ is δ-close to a degree-d polynomial. It uses the notion of a *line*.

Definition 4.11: A *line* in F^m is a set q points with a parametric representation of the form $\{\hat{u}_0 + t \cdot \hat{u}_1 : t \in \mathrm{F}\}$ for some $\hat{u}_0, \hat{u}_1 \in \mathrm{F}^m$. (The symbol $+$ denotes componentwise addition of two vectors of length m.) Note that it is identical to the line $\{\hat{u}_0 + t \cdot c\hat{u}_1 : t \in \mathrm{F}\}$ for any $c \in \mathrm{F} \setminus \{0\}$. Our convention is to fix one of the representations as canonical.

Definition 4.12: Let l be a line in F^m whose canonical representation is $\{\hat{u}_0 + t \cdot \hat{u}_1 : t \in \mathrm{F}\}$, and $g : \mathrm{F} \to \mathrm{F}$ be a function of one variable, t. The *value produced by g at the point $\hat{u}_0 + a \cdot \hat{u}_1$* of l is $g(a)$.

Note that if a function f is in $\mathrm{F}_d[x_1, \ldots, x_m]$, then the values of f on any line are described by a univariate degree-d polynomial in parameter t. We illustrate this fact with an example.

Example 4.4: Let $f \in \mathrm{F}_4[x_1, x_2]$ be a bivariate polynomial of degree 4 defined as $f(x_1, x_2) = x_1 x_2^3 + x_1^2$. Consider the line $\{(a_1, a_2) + t \cdot (b_1, b_2) : t \in \mathrm{F}\}$. Define a function $h : \mathrm{F} \to \mathrm{F}$ as

$$h(t) = (a_1 + b_1 t)(a_2 + b_2 t)^3 + (a_1 + b_1 t)^2.$$

It is a univariate polynomial of degree 4, and describes f at every point on the line. For instance, the value produced by h at the point (a_1, a_2) is $h(0) = a_1 a_2^3 + a_1^2 = f(a_1, a_2)$.

The Low-degree Test picks lines uniformly at random from all lines in F^m, and checks how well the restriction of f is described by a univariate polynomial. For purposes of efficiency it requires that a table T be provided along with f, supposedly containing for each line the best univariate degree d polynomial describing f on that line. It performs many repetitions of the following trial: Pick a line uniformly at random and a point uniformly at random on this line. Read the univariate polynomial provided for the line in the table, and check whether it describes f at the point.

Low-degree Test, Procedure 4.2-(i)

Inputs: $f : \mathrm{F}^m \to \mathrm{F}$, $\delta < 10^{-4}$.
To Verify: f is δ-close to $F_d[x_1, \ldots, x_m]$.
Given : A table T containing, for each line l, a univariate degree d polynomial P_l.

Pick $k = 4/\delta$ random lines $l_1, \ldots, l_k$ and a random point on each of these lines $z_1 \in_R l_1, \ldots, z_k \in_R l_k$.
Read $P_{l_1}, \ldots, P_{l_k}$ and $f(z_1), \ldots, f(z_k)$.
If for $i = 1, \ldots, k$, P_{l_i} correctly describes f at z_i
ACCEPT
else
REJECT.

Complexity: Generating a random line requires picking the line parameters $\hat{u}_0, \hat{u}_1$ randomly from F^m. This requires only $2m \log q$ random bits. Also, the test reads only $4/\delta$ values of f and the same number of line polynomials.

Correctness: Clearly, if f is a degree-d polynomial, then by making the table T contain the univariate polynomials that describe the lines in F^m, the test can be made to accept with probability 1. The next theorem shows that if f is not δ-close, then the test rejects with high probability irrespective of the contents of the table.

Theorem 4.15: *Let the field size q be $\Omega(d^3 m)$. If f is not δ-close, for $\delta < 10^{-4}$, then the low-degree test accepts with probability* $< 1/4$.□

We defer the proof of Theorem 4.15 until Chapter 5.

Next, we describe Procedure 4.2-(ii). This procedure, given a δ-close function $f : \mathrm{F}^m \to \mathrm{F}$ (where δ is a sufficiently small constant) and a sequence of l points $z_1, \ldots, z_s$, recovers the values $\widetilde{f}(z_1), \ldots, \widetilde{f}(z_s)$. (The procedure sometimes produces erroneous output with some small probability.) The procedure uses the notion of a *curve*, whose definition generalizes that of a line.

Definition 4.13: A *degree-k curve* in F^m is a set of q points with a parametric representation of the form

$$\left\{\hat{u}_0 + t \cdot \hat{u}_1 + \cdots + t^k \hat{u}_k : t \in \mathrm{F}\right\}$$

where $\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_k \in \mathrm{F}^m$ (the symbol $+$ denotes componentwise addition of two vectors of length m). Note that it is identical to the curve

$$\left\{\hat{u}_0 + t \cdot c\hat{u}_1 + \cdots + t^k \cdot c^k \hat{u}_k : t \in \mathrm{F}\right\}$$

for any $c \in \mathrm{F} \setminus \{0\}$. Our convention is to fix one of the representations as canonical.

Definition 4.14: Let C be a degree-k curve in F^m whose canonical representation is $\left\{\hat{u}_0 + t \cdot \hat{u}_1 + \cdots + t^k \hat{u}_k : t \in \mathrm{F}\right\}$, and $g : \mathrm{F} \to \mathrm{F}$ be a function of one variable, t. The *value produced by* g *at the point* $\hat{u}_0 + a \cdot \hat{u}_1 + \cdots + a^k \cdot \hat{u}_k$ *of* C is $g(a)$.

Note that if a function f is in $\mathrm{F}_d[x_1, \ldots, x_m]$, then the values of f on any degree-k curve are described by a univariate degree-dk polynomial in parameter t.

In describing the procedure, we assume that elements of field F are ordered in some canonical fashion. Thus we can talk about the ith point of a curve, for any positive integer i less than $q + 1$. Furthermore, when we refer to the "set of points on a curve," we are actually referring to a multiset, since a curve could pass through the same point more than once. Having clarified this, we observe next that a degree k curve is fixed once we know its first $k + 1$ points.

Fact 4.16: *For any set of* $k + 1$ *points* $z_1, \ldots, z_{k+1} \in F^m$ *(where there could be repetitions among* $z_1, \ldots, z_{k+1}$*), there is a unique degree* k *curve whose first* $k+1$ *points are* $z_1, \ldots, z_{k+1}$*.*

Proof: To specify a degree-k curve $\left\{\hat{u}_0 + t \cdot \hat{u}_1 + \cdots + t^k \hat{u}_k : t \in \mathrm{F}\right\}$ in F^m, we need to specify its coefficients $\hat{u}_0, \ldots, \hat{u}_k$, which are m-tuples over F. Clearly, an equivalent specification consists of m univariate polynomials $g_1, \ldots, g_m$ in the curve parameter t. Each g_i has degree at most k, and the tth point on C is $(g_1(t), \ldots, g_m(t)) \in F^m$.

Fact A.3 (Appendix A) implies that once we fix the first $k+1$ points of C, the polynomials $g_1, \ldots, g_m$ are uniquely determined. Hence the curve is also uniquely determined. $\square$

Definition 4.15: For any points $z_1, \ldots, z_k$ in F^m, let $P(< z_1, \ldots, z_k >)$ denote the set of degree-k curves whose first k points are $z_1, \ldots, z_k$.

Let $z_1, \ldots, z_s$ be the points at which the procedure has to reconstruct values of $\widetilde{f}$. Since $s + 1$ points fix a degree-c curve, the number of curves in $P(< z_1, \ldots, z_s >)$ is q^m. The procedure requires a table containing, for each such curve, a degree-sd polynomial that best describes f on the curve. It picks a curve uniformly at random from the set and reads the polynomial corresponding to it in the table. It checks that the polynomial agrees with f on a random point on the curve. As we will show, this provides good confidence that the polynomial correctly describes $\widetilde{f}$, and therefore the procedure outputs the values taken by this polynomial at $z_1, \ldots, z_s$.

Extracting s values of $\widetilde{f}$; Procedure 4.2-(ii)

Input: A function $f : \mathrm{F}^m \rightarrow \mathrm{F}$ that is δ-close to $\mathrm{F}_d[x_1, \ldots, x_m]$, and c points $z_1, \ldots, z_s \in \mathrm{F}^m$.
Aim: To obtain $\widetilde{f}(z_1)$, $\widetilde{f}(z_2), \ldots, \widetilde{f}(z_s)$.
Given: A table T containing, for each degree-c curve C passing through $z_1, \ldots, z_s$, a univariate polynomial T_C in the parameter t of degree ds.

Pick a degree-s curve C randomly from $P(<z_1, \ldots, z_s>)$.
Pick a point y randomly from the set of points in C.
Look up the polynomial $T_C(t)$ and the value $f(y)$.
If T_C correctly describes f at y
then
output the values produced by $T_C(t)$ at $z_1, \ldots, z_s$ as $\widetilde{f}(z_1), \ldots, \widetilde{f}(z_s)$
else
REJECT

Note that if $f \in \mathrm{F}_d[x_1, \ldots, x_m]$ and the table contains curve polynomials that describe f correctly, the procedure will never output REJECT, and correctly output $\widetilde{f}(z_1)$, $\widetilde{f}(z_2), \ldots, \widetilde{f}(z_s)$.

Lemma 4.17: *If f is δ-close, then*

$$\Pr[\textit{the procedure outputs values that are not } \widetilde{f}(z_1), \ldots, \widetilde{f}(z_s)] < 2\sqrt{\delta} + o(1),$$

no matter what the table of curve polynomials contains.

Proof: For a curve C let us denote by $\widetilde{f}|_C(t)$ the univariate polynomial in t of degree sd that describes $\widetilde{f}$ on points in C. The only case in which our procedure can output incorrect values of $\widetilde{f}$ is when it picks a curve C such that the polynomial $T_C(t)$ provided in the table is different from $\widetilde{f}|_C(t)$. We will show that on most curves such an incorrect $T_C(t)$ does not describe f on most points of the curve. Hence the procedure outputs REJECT with high probability when it compares the values of $T_C(t)$ and f on random point y of the curve.

First, we state a claim, which we will prove later.

Claim: Let $\gamma = 1 - \sqrt{\delta}$. For at least γ fraction of curves in $P(< z_1, \ldots, z_s >)$ the following is true. Let C denote the curve. Then

$$\widetilde{f}|_C(t) \text{ describes } f \text{ at } \gamma \text{ fraction of points in } C. \tag{4.15}$$

Let C be a curve that satisfies the condition in (4.15). Recall that two different degree-ds univariate polynomials agree at no more than ds points in F. Hence if $\widetilde{f}|_C$ describes f on γ

fraction of points in the curve, then every other univariate polynomial of degree d describes f in no more than $1-\gamma-ds/q$ fraction of points. In particular, suppose $T_C \neq \tilde{f}|_C$. Then T_C describes f at no more than $1-\gamma+cd/q=\sqrt{\delta}+ds/q$ fraction of points in C.

Hence an upperbound on the probability of outputting incorrect values of $\tilde{f}$ is

$$\Pr_C[C \text{ does not satisfy condition (4.15)}] + \sqrt{\delta} + \frac{ds}{q},$$

which is at most $1-\gamma+\sqrt{\delta}+\frac{ds}{q}=2\sqrt{\delta}+o(1)$.

Hence the lemma is proved. Now we prove the claim.

Proof of the claim: Let $S \subseteq \mathrm{F}^m$ be the set of points where functions f and $\tilde{f}$ disagree. Then S constitutes a fraction at most δ of F^m. Lemma A.9 implies that when we pick a curve randomly from $P(<z_1,\ldots,z_k>)$, the expected fraction of points on the curve that are in S is $\frac{|S|}{|\mathrm{F}|^m}$, which is at most δ. The Averaging Principle (Fact A.2) implies that the fraction of curves on which

$$\text{more than } \sqrt{\delta} \text{ fraction of points of the curve are in } S$$

is no more than $\sqrt{\delta}$.

This proves the claim, and hence Lemma 4.17. □

4.5. The Overall Picture

Our proof of the PCP Theorem consists in defining two verifiers (namely, the ones in Theorems 3.2 and 3.4), and composing them (using the Composition Lemma) to construct a $(\log n, 1, 1)$-restricted verifier.

While describing each verifier we described how to encode a satisfying assignment, such that the verifier accepts the encoding with probability 1. When we compose two verifiers, their associated encoding schemes get composed as well (where "composition" has to be defined appropriately using the construction in the proof of the Composition Lemma).

In other words, our proof of the PCP Theorem basically consists of a (complicated) definition of an encoding scheme, itself defined as a composition of other schemes. Figure 4.3 gives a bird's-eye view of different steps in the encoding scheme. Each step corresponds to a different verifier. Next to each verifier we have written down the parameters associated with it: the number of random bits, the number of queries, and the decision time.

4.6. History/Attributions

The techniques in this section are inspired by the phenomenon of *random-self-reducibility*. A function f (on domain say, $\{0,1\}^n$) is said to exhibit this phenomenon if the computation of f on a worst-case input, x, can be reduced in polynomial time to the computation of f on a small number of inputs that are uniformly (not necessarily independently) distributed in $\{0,1\}^n$.

The phenomenon in some sense underlies cryptography. For example, the pseudo-random generator of [BM84] uses the fact that the discrete log problem is random self-reducible. The term *random-self-reducible* appears to have been explicitly defined first in [AFK89] (see also [FKN90]). Many flavors of random-self-reducibility have since been defined and studied ([FF93, Fei93]).

Blum ([BK89]) and Lipton ([Lip89]) rephrased the r.s.r. property as follows. Given any program that computes f on "most" inputs, a randomized algorithm can recover the value of f at an arbitrary input x. This observation was the starting point for the theory of self-testing/self-correcting programs ([BK89, Lip89, BLR90]). Most functions to which this theory seemed to apply were algebraic in nature (see [Yao90] for some nonalgebraic examples, though).

A new insight was made in [BF91, Sha92, LFKN92]: it is possible to represent logical formulae using low degree polynomials. But recall that different classes of logical formulae are complete for complexity classes like PSPACE, NEXPTIME, NP etc.. This at once suggests that randomized techniques from program-testing/correcting should be applicable to the study of conventional complexity classes. The results in this chapter are precisely of this character.

Techniques of this chapter: attributions. The first algebraic representation of 3SAT is due to [BFL91]. That representation (using multilinear polynomials) has a problem: the field-size and number of variables required to represent a string of n bits are such that generating a random point in the space F^m requires more than $O(\log n)$ random bits. The polynomial extension used in this chapter does not suffer from this problem. It is due to ([BFLS91]). The Sum-check procedure (Procedure 4.3) is due to [LFKN92].

Thus Lemma 4.5 could be proved with minor modifications of the above-mentioned results, although no previous paper had proved it explicitly before [AS92]. All other results in this chapter, except the Low-degree Test and the idea of checking split assignments, are from [ALM$^+$92]. The idea of checking split assignments is from [AS92]. The history of the Low-degree Test will be covered in the next section.

The discovery of the proof of Lemma 4.4 was influenced by the *parallelization* procedures of [LS91, FL92], although the ideas used in this lemma can be traced back to [BF90, LFKN92]. The design of the verifier of Section 4.3 owes much to existing examples of self-testing/correcting programs from [BLR90, Fre79].

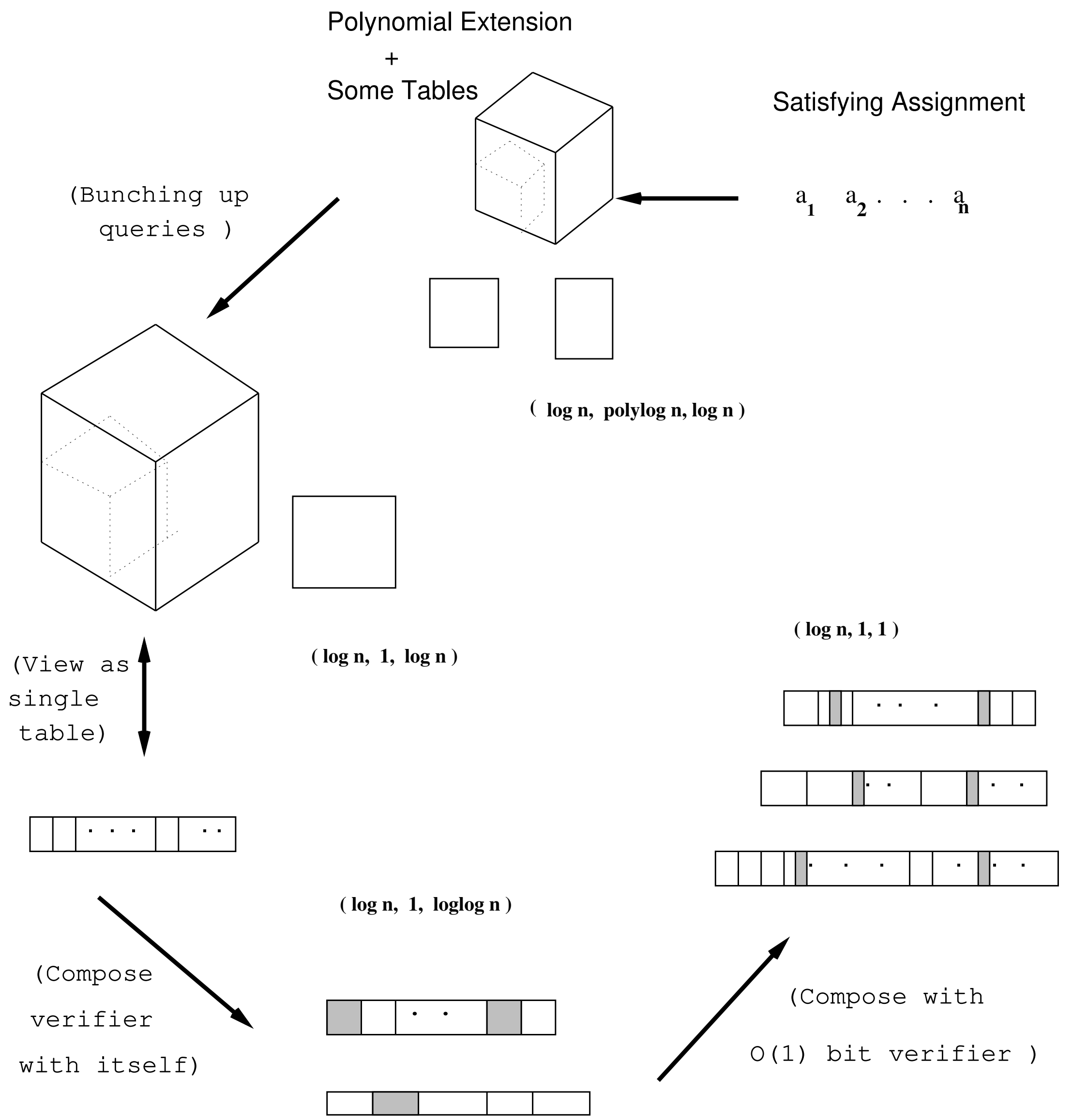

Figure 4.3: How to encode an assignment so that the PCP($\log n, 1$) verifier accepts with probability 1.

Chapter 5

The Low-degree Test

This chapter contains a proof of the correctness of the Low-degree Test, specifically, a proof of Theorem 4.15.

Let d be an arbitrary integer that is fixed for the rest of the chapter. Let F be the finite field $\mathrm{GF}(q)$, where q might depend on d. We continue to use $\mathrm{F}_d[x_1, \ldots, x_m]$, the code of m-variate polynomials of degree d, which was defined in Section 4.1. We will also use properties of lines in F^m, which were defined in Section 4.4.3 (Definitions 4.11 and 4.12).

We remind the reader of the following observation from Section 4.4.3 (specifically, Example 4.4): If a function f is in $\mathrm{F}_d[x_1, \ldots, x_m]$, then the values of f on any line are described by a univariate degree-d polynomial in the line parameter t. The Low-degree Test is based on a strong contrapositive of that observation: If on "most" lines, "most" values of f are described by a univariate polynomial of degree d, then f itself is ϵ-close to $\mathrm{F}_d[x_1, \ldots, x_m]$ for some small ϵ. To state this contrapositive more precisely we need to define some concepts.

Definition 5.1: Let $f : \mathrm{F}^m \to \mathrm{F}$ be a function and l be a line. The symbol P_l^f denotes the univariate degree d polynomial (in the line parameter t) that describes f on more points of l than any other degree d polynomial. (We arbitrarily break ties among different polynomials that describe f equally well on the line.)

Note: (i) To make the degree d explicit, we could have used the symbol $P_d^f(l)$ instead of P_l^f. But the degree d can always be inferred from the context. (ii) Suppose line l is such that polynomial P_l^f describes f on more than $1/2 + d/2q$ fraction of points in l. Then no other univariate polynomial can do better. (For, if any other polynomial does as well as P_l^f, then there is a set of at least $d+1$ points on which they both describe f. But if two degree d polynomials agree on $d+1$ points, they are the same.) In other words, for such a line l, the best line polynomial P_l^f is uniquely-defined.

Now we introduce some notation. Let S be any finite set and let μ be a real-valued function defined on S. The *average value of* μ *on* S is denoted by $\mathrm{E}_{x \in S}[\mu(x)]$. We justify this notation on the grounds that if we pick x uniformly at random from S, the expectation of $\mu(x)$ is exactly the average value of μ on S. (When S is clear from context, we drop S and use $E_x[\mu(x)]$.)

Definition 5.2: Let $f : \mathrm{F}^m \to \mathrm{F}$ be a function and l be a line. The *success rate of f on l*, denoted $\mu^f(l)$, is defined as

$$\mu^f(l) = \text{fraction of points on } l \text{ where } P_l^f \text{ describes } f.$$

The *success rate* of f is defined as the average success rate of f among all lines, that is, as $\mathrm{E}_{l\in L}[\mu^f(l)]$ where L is the set of lines in F^m.

The following theorem is the precise statement of the contrapositive mentioned above. It is proved in Section 5.2. A crucial component in its proof is Lemma 5.2.

Theorem 5.1: *Let d, m be integers satisfying $100d^3m < q$. Every function $f : F^m \to F$ whose success rate is $(1-\alpha)$, for $\alpha \leq 10^{-4}$, is 2α-close to $F_d[x_1, \ldots, x_m]$.*

(The proof can be tightened so that it works for larger α. Whether it works for $\alpha \geq 0.5$ (say) is open.)

Now we prove that the Low-degree Test works as claimed.

Proof: (*Of Theorem 4.15*) Let $f : \mathrm{F}^m \to \mathrm{F}$ be a function and T be a table of line polynomials provided to the Low-degree Test. Let p_T denote the success probability of the following trial: Pick a random line l and a random point $y \in l$. Call the trial a success if the polynomial provided for l in table T correctly describes f at x.

By definition, the success rate of f is the maximum, over all tables T, of p_T. Hence if f is not δ-close, where $\delta \leq 10^{-4}$, then Theorem 5.1 implies that $p_T < 1 - \delta/2$ for every table T.

Recall that the Low-degree Test performs $4/\delta$ repetitions of the above trial, and accepts iff all succeed. Now suppose f is not δ-close. After repeating the trial $4/\delta$ times, the test will reject with probability at least $1/2$. Thus Theorem 4.15 is proved. $\Box$

The proof of Theorem 5.1 relies on the following Lemma, which asserts the truth of the theorem statement when the number of variables, m, is 2. The lemma is proved in Section 5.1.

Lemma 5.2: *Let integer d be such $q = \Omega(d^3)$. Every $g : F^2 \to F$ whose success rate is $1-\beta$, for $\beta < 0.001$, is $10\sqrt{\beta}$-close.*

Convention: A typical lemma statement in this chapter says "If the hypothesis $\mathcal{H}$ holds for a number β where β is small enough, then the conclusion $\mathcal{C}$ holds for the number c_β," where c_β is explictly defined in terms of β. (In Lemma 5.2 for example, c_β is $10\sqrt{\beta}$.) When applying the lemma later we will need to make c_β smaller than some suitably small constant, which we can do by making β some other suitably small constant. Having clarified this, from now on we will never state explicit values of such constants.

5.1. The Bivariate Case

This section contains a proof of Lemma 5.2.

For an element $a \in \mathrm{F}$ let the the *row* $\{y = b\}$ be the line of points in F^m whose second coordinate is b, i.e., the set $\{(x, b) : x \in \mathrm{F}\}$. The *column* $\{x = a\}$ for $a \in \mathrm{F}$ is defined similarly. (We often use the shorthand "row b" for "row $\{y = b\}$.") As usual, $P^f_{\{y=b\}}$ is the the degree-d polynomial (in x) that best describes f in the row $\{y = b\}$. For clarity we denote it by R_b. Likewise, C_a denotes a degree-d polynomial (in y) that best describes f on the column $\{x = a\}$. The following lemma says that if on most points in F^2 the row and column polynomial describe f correctly, then f is ϵ-close for some small ϵ. (At the end of this section we derive Lemma 5.2 easily from this lemma.)

Lemma 5.3: *Let $f : F^2 \to F$ be a function. Suppose at more than fraction $1 - \delta$ of points $(a, b) \in F^2$ we have*

$$f(a, b) = R_b(a) = C_a(b),$$

where δ is small enough. Then f is $5\sqrt{\delta}$-close to a polynomial of bidegree (d, d).

Let a function be called a *rational polynomial* if it can be expressed as h/g, where h and g are polynomials.

We divide our goal of proving Lemma 5.3 into two subgoals.

Subgoal 1: *Show that there are bivariate polynomials h, g with fairly low degree, such that on most points in F^2, function f agrees with the rational polynomial h/g.*

Subgoal 2: *Show that the rational polynomial $h(x, y)/g(x, y)$ obtained from Subgoal 1 is a bivariate polynomial of degree d.*

Why is Subgoal 2 realistic? Using the hypothesis of Lemma 5.3 and an averaging argument, we can show that f's restriction to most rows and columns is ϵ-close, for some small ϵ. Further, the statement of Subgoal 1 says that f and h/g agree on most points. Another use of averaging shows that the restriction of h/g on most rows and columns is ϵ'-close, for some ϵ'. A simple argument, using Euclid's division algorithm for polynomials, will then show that h/g is a bivariate polynomial of degree d. (For details see the proof of Lemma 5.7.)

Why is Subgoal 1 realistic? We give a motivating example from the univariate case.

Example 5.1 ([BW]): Let k be an integer much bigger than d. Let (a_1, b_1), (a_2, b_2), $\ldots, (a_k, b_k)$ be a sequence of pairs of the type (point, value), where the points a_i are in F and so are the values b_i. Further, suppose there is some univariate polynomial r of degree d that describes 3/4th of the pairs, that is,

$$r(a_i) = b_i \quad \text{for } \tfrac{3k}{4} \text{ values of } i \in \{1, 2, \ldots, k\}.$$

We show how to construct univariate polynomials c, e of degree at most $\frac{k}{4} + d$ and $\frac{k}{4}$ respectively, such that

$$c\,(a_i) = b_i\ e(a_i) \quad \forall i \in \{1, \ldots, k\}\,.$$

In other words, we construct a rational polynomial $c(x)/e(x)$ that describes the entire sequence of pairs.

Let e be a non-zero univariate polynomial of degree $k/4$ that is 0 on the set $\{a_i : r(a_i) \neq b_i\}$. (Fact A.3 implies that such a polynomial e exists.) Then we have

$$e(a_i) r(a_i) = b_i\ e(a_i) \quad \forall i \in \{1, \ldots, k\}\,.$$

By defining c as $c(x) = e(x)r(x)$, we're done. □

Let us consider the relevance of Example 5.1 to Subgoal 2. Recall that the hypothesis of Lemma 5.3 says that at a "typical" point in F^2, the row polynomial, the column polynomial, and f all agree. Pick k "typical" columns, say $a_1, \ldots, a_k$. Pick a "typical" row, $\{y = c\}$. Denote by $b_1, b_2, \ldots, b_k$ the values taken by the column polynomials $C_{a_1}(y), \ldots, C_{a_k}(y)$ respectively in this row (that is, b_i is $C_{a_i}(c)$). Since we chose typical columns and row, the row polynomial R_c should describe most of these values correctly. For argument's sake, suppose it describes 3/4th of the values, that is

$$R_c(a_i) = b_i \quad \text{for } \tfrac{3k}{4} \text{ values of } i \in \{1, 2, \ldots, k\}.$$

Then the construction of Example 5.1 applies to the sequence of pairs (a_1, b_1), (a_2, b_2), $\ldots$, (a_k, b_k), and we can construct a low-degree rational polynomial $c(x)/e(x)$ that describes all of $b_1, \ldots, b_k$.

But there is an added twist. The values $b_1, \ldots, b_k$ are not any arbitrary elements of F. Each is itself the value of the corresponding column polynomial, which is a degree d polynomial in a separate parameter, y. Hence a more proper way to view the b_i's in this case is as elements of $F[y]$, the domain of polynomials in the formal variable y. This viewpoint, along with the fact that a good fraction of rows are "typical," allows us to find a good description of f in terms of a rational bivariate polynomial.

Now we state the results precisely. First we make explicit the algebraic object that underlies Example 5.1: an overdetermined system of linear equations.

Lemma 5.4: *Let $\{(a_1, b_1), \ldots, (a_{36d}, b_{36d})\}$ be a sequence of $36d$ (point, value) pairs for which there is a univariate degree-d polynomial r such that*

$$r(a_i) = b_i \quad \textit{for } 27d \textit{ values of } i \in \{1, 2, \ldots, 36d\}.$$

Then the following system of equations (over F) in the variables $(c_0, \ldots c_{10d},\ e_0,\ \ldots, e_{9d})$ has a nontrivial solution.

$$\begin{aligned} c_0 + c_1 a_1 + \ldots + c_{10d} a_1^{10d} &= b_1 \cdot (e_0 + e_1 a_1 + \cdots + e_{9d} a_1^{9d}) \\ c_0 + c_1 a_2 + \ldots + c_{10d} a_2^{10d} &= b_2 \cdot (e_0 + e_1 a_2 + \cdots + e_{9d} a_2^{9d}) \\ &\vdots \\ c_0 + c_1 a_{36d} + \ldots + c_{10d} a_{36d}^{10d} &= b_{36d} \cdot (e_0 + e_1 a_{36d} + \cdots + e_{9d} a_{36d}^{9d}) \end{aligned}$$

Proof: As in Example 5.1, construct polynomials c, e of degree $10d$ and $9d$ respectively such that

$$c(a_i) = e(a_i) b_i \quad \forall i \in \{1, 2, \ldots, 36d\}.$$

W.l.o.g. we assume that at least one of the polynomials e, c is not the zero polynomial.

Now let e and c be expressed as $e(x) = \sum_{i=0}^{9d} e_i x^i$, and $c(x) = \sum_{j=0}^{10d} e_j x^j$ respectively. Then $c_0, \ldots, c_{10d}, e_0, \ldots, e_{9d}$ is a nontrivial solution to the given linear system. $\Box$

Note: The linear system in the statement of Lemma 5.4 is overdetermined: it consists of $36d$ equations in $19d + 2$ variables. Represent the system in standard form as $A \cdot \overline{x} = 0$, where $\overline{x}$ is the vector of variables $(c_0, c_1, \ldots, c_{10d}, e_0, \ldots, e_{9d})$. The system has a nontrivial solution. Hence Fact A.8-(i) of the Appendix implies that the determinant of every $(19d + 1) \times (19d + 1)$ submatrix of A is 0.

Definition 5.3: If l and m are integers, a bivariate polynomial g in the variables x and y *has bidegree* (l, m) if its degree in x and y is l and m respectively.

The following Lemma achieves Subgoal 1.

Lemma 5.5: *Let f be the function of Lemma 5.3. Then there are polynomials $g(x, y)$, $h(x, y)$ of bidegree $(9d, 20d^2)$, $(10d, 20d^2)$ respectively such that $h(x, y)/g(x, y)$ describes f in $1 - \gamma$ fraction of the points, where $\gamma < 5\sqrt{\delta}$.*

Proof: The hypothesis of Lemma 5.3 implies that in the average column, the column polynomial describes f (and the row polynomials) in a fraction $1 - \delta$ of the the points. Since $36d = o(|\mathrm{F}|)$, an averaging argument shows the following. There are $36d$ columns $a_1, \ldots,$ and a_{36d} such that for $i = 1, \ldots, 36d$, the column polynomial $C_{a_i}(y)$ describes f at more than $1 - \delta - o(1)$ fraction of the points in the column a_i.

For counting purposes, we put a $\star$ at every point in these columns at which the row and column polynomials agree. Let y be a row that has at least $27d$ $\star$'s. Then the sequence of pairs $(a_1, C_{a_1}(y)), \ldots, (a_{36d}, C_{a_{36d}}(y))$ satisfy the hypothesis of Lemma 5.4. Hence the following system of equations has a solution.

$$\begin{aligned}
c_0 + c_1 a_1 + \ldots + c_{10d} a_1^{10d} &= C_{a_1}(y) \cdot (e_0 + e_1 a_1 + \cdots + e_{9d} a_1^{9d}) \\
c_0 + c_1 a_2 + \ldots + c_{10d} a_2^{10d} &= C_{a_2}(y) \cdot (e_0 + e_1 a_2 + \cdots + e_{9d} a_2^{9d}) \\
&\vdots \\
c_0 + c_1 a_{10d} + \ldots + c_{20d} a_{36d}^{10d} &= C_{a_{10d}}(y) \cdot (e_0 + e_1 a_{10d} + \cdots + e_{9d} a_{36d}^{9d})
\end{aligned}$$

Represent the system in standard form as $A \cdot \overline{x} = 0$, where $\overline{x}$ is the vector of variables $(c_0, c_1, \ldots, c_{10d}, e_0, \ldots, e_{9d})$, and A is the $36d \times (19d+2)$ matrix

$$\begin{pmatrix}
1 & a_1 & a_1^2 & \cdots & a_1^{10d} & -C_{a_1}(y) & -a_1 C_{a_1}(y) & \cdots & -a_1^{9d} C_{a_1}(y) \\
1 & a_2 & a_2^2 & \cdots & a_2^{10d} & -C_{a_2}(y) & -a_1 C_{a_2}(y) & \cdots & -a_2^{9d} C_{a_2}(y) \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\
1 & a_{36d} & a_{36d}^2 & \cdots & a_{36d}^{10d} & -C_{a_{36d}}(y) & -a_{36d} C_{a_{36d}}(y) & \cdots & -a_{36d}^{9d} C_{a_{36d}}(y)
\end{pmatrix}.$$

We take two different views of this system. In the first view it is a family of q separate systems, one for each value of y in F. The system corresponding to $y = b$ has a solution iff the number of $\star$'s in the row is $27d$. Averaging implies there are $q(1 - \frac{\delta}{3/4})$ such values of y. Further, upon substituting any such value of y in A, the determinant of every $(19d+1) \times (19d+1)$ submatrix of A becomes 0. (See the note following Lemma 5.4.)

In the second view, A is the matrix of a system of equations over the domain F$[y]$, the domain of polynomials in the formal variable y. We show that we can find a non-trivial solution to $c_0, \ldots, c_{10d}, e_0, \ldots, e_{9d}$ in this domain. More specifically, we show that the determinant of every $(19d+1) \times (19d+1)$ submatrix of A is the zero polynomial of F$[y]$, and then use Fact A.8, part 1.

Let B be any $(19d+1) \times (19d+1)$ sub-matrix of A. Fact A.7 implies that $det(B)$, the determinant of B, is a polynomial of degree at most $19d+1$ in the entries of B. By inspection, each entry of B is a polynomial in y of degree at most d. Therefore $det(B)$ is a polynomial in y of degree at most $(19d+1)d$. Either $det(B)$ is the zero polynomial, or it is has at most $(19d+1)d$ roots. But as already noted, the determinant of every $(19d+1) \times (19d+1)$ submatrix of A becomes 0 for at least $q(1 - \frac{\delta}{3/4})$ values of y. Since $q(1 - \frac{\delta}{3/4}) > (19d+1)d$, we conclude $det(B)$ is the zero polynomial.

We have shown that the system has a nontrivial solution in the domain F$[y]$. Further, Fact A.8, part 2, implies that a non-trivial solution can be found in which each of $c_0, \ldots, c_{10d}, e_0, \ldots, e_{9d}$ is itself a polynomial of degree $19d+1$ in the entries of A. In other words, each e_i and c_i is a polynomial in y of degree $(19d+1)d$. Let $c_0, \ldots, c_{10d}, e_0, \ldots, e_{9d} \in$

$F_{20d^2}[y]$ be such a solution. Define the bivariate polynomials h, g as

$$h(x,y) = \sum_{i=0}^{10d} c_i(y)x^i \quad \text{and} \quad g(x,y) = \sum_{i=0}^{9d} e_i(y)x^i.$$

Clearly, h, g have bidegree $(10d, 20d^2)$, and $(9d, 20d^2)$ respectively.

It only remains to show the following.

Claim 1: The rational polynomial h/g describes f in $1 - 5\sqrt{\delta}$ fraction of points in F^2.

To this end we prove Claim 2.

Claim 2: A $(1 - \sqrt{\delta} - 4\delta - o(1))$ fraction of rows satisfy all the following properties. (i) The restriction of g to this row is not the zero polynomial in $\mathrm{F}[x]$. (ii) The row contains at least $10d$ $\star$'s. (iii) The row polynomial describes f at a $1 - \sqrt{\delta}$ fraction of points of the row.

First we show that Claim 2 implies Claim 1.

Let $\{y = b\}$ be a row that satisfies conditions (i) and (ii) in the statement of Claim 2. Then the univariate polynomial $h(x,b) - R_b(x)g(x,b)$ has degree $10d$ but more than $10d$ roots. Hence the polynomial is identically zero, in other words, $h(x,b)/g(b,x) = R_b(x)$. Now if the row also satisfies (iii), then h/g describes f on at least $(1-\sqrt{\delta})$ fraction of points in this row.

Thus Claim 2 implies that h/g describes f on at least $(1 - \sqrt{\delta} - 4\delta - o(1))(1 - \sqrt{\delta})$ fraction of points in F^2. This fraction is at least $1 - 5\sqrt{\delta}$. Hence Claim 1 follows.

Now we prove Claim 2.

Averaging shows that at most $\sqrt{\delta}$ fraction of the rows fail to satisfy (iii).

The polynomial $g(x,y)$ is non-zero, and its degree in y is at most $20d^2$. Hence at most $20d^2$ rows fail to satisfy (i).

Finally, in each of columns $a_1, \ldots, a_{36d}$ a $(1 - \delta - o(1))$ fraction of points contain $\star$'s. Averaging shows that the fraction of rows that fail (ii) is at most $\frac{\delta}{10/36}$, that is, at most 4δ.

Hence the fraction of rows that satisfy all of (i), (ii) and (iii) is at least $1 - 20d^2/q - 4\delta - \sqrt{\delta}$. Since $d^2/q = o(1)$, Claim 2 follows.

□

Next, we move towards Subgoal 2.

Lemma 5.6: *If r, s, t are univariate polynomials in x of degree l, m, n respectively, and*

$$r(x) = s(x)\,t(x) \quad \textit{for} \ \ \max\{l, m+n\} \ \textit{values of } x$$

then $r = s \cdot t$.

Proof: The univariate polynomial $r - s \cdot t$ has degree $\max\{l, m+n\}$. If it has more than $\max\{l, m+n\}$ roots, it is identically zero. □

Note: Let r, s be univariate polynomials of degree l, m respectively. Assume $\max\{l, m\}$ is $\leq q\epsilon$ for some $\epsilon > 0$. Lemma 5.6 implies that the rational polynomial r/s can only exhibit two behaviors. Either it is a univariate polynomial of degree $l - m$, or else it is not even ϵ-close to any univariate polynomial of degree $l - m$.

The following lemma achieves Subgoal 2.

Lemma 5.7: *If h, g are polynomials obtained in Lemma 5.5, then h/g is a bivariate polynomial of bidegree (d, d).*

Proof: The proof consists in three claims.

The first claim is that the restriction of h/g on most rows and columns is a univariate polynomial of degree d. This uses two observations. First, the restriction of h/g to most rows and columns describes f quite well, and is therefore a α-close for some small α. Second, Lemma 5.6 implies that if the restriction of h/g on any row or column is α-close, then the restriction is a univariate polynomial.

Claim 1: For at least 1/2 the elements $a \in \mathrm{F}$, $h(x, a)/g(x, a)$ is 0.5-close (and hence by Lemma 5.6 is in $\mathrm{F}_d[x]$).

Indeed, by averaging, in at least $1 - \sqrt{\delta}$ of the rows, f is $\sqrt{\delta}$-close. Let γ be as in Lemma 5.5. Then in at least $1 - \sqrt{\gamma}$ fraction of the rows $h(x, y)/g(x, y)$ describes f in $1 - \sqrt{\gamma}$ fraction of points. Hence in $1 - \sqrt{\delta} - \sqrt{\gamma}$ fraction of rows the restriction of $h(x, y)/g(x, y)$ is $(\sqrt{\delta} + \sqrt{\gamma})$-close. Since $\sqrt{\delta} + \sqrt{2\gamma} < 0.5$, so Claim 1 is proved.

Let k, l stand for $20d^2, 9d$ respectively, so that the bidegrees of g, h are (l, k) and $(l+d, k)$ respectively.

The second claim is that the degree of h/g in x is d.

Claim 2: Assuming Claim 1, there exists a polynomial $s(y)$ of degree dk such that hs/g is a polynomial of bidegree $(d, 20d^3)$.

Assume w.l.o.g. that h, g have no common factor. Represent $g(x, y), h(x, y)$ as $\sum_{i=0}^{l} p_i(y)x^i$ and $\sum_{i=0}^{l+d} s_i(y)x^i$ respectively where $p_0(y), \ldots, p_l(y)$, and $s_0(y), \ldots, s_{l+d}(y)$ are univariate polynomials of degree k. Using Euclidean pseudo-division for polynomials (see [Knu74], p. 407), we obtain polynomials $q(x, y)$, $r(x, y)$ of bidegree $(d, k + dl)$ and $(l - 1, k + dl)$ respectively such that

$$(p_l(y))^d h(x, y) = q(x, y)g(x, y) + r(x, y).$$

For every $a \in \mathrm{F}$ such that $g(x,a)|h(x,a)$, it must be that $r(x,a)$ is a zero polynomial. By Claim 1, at least 1/2 the points $a \in \mathrm{F}$ have this property. Hence $r(a,x)$ is the zero polynomial for at least $q/2$ values of a, which is more than $ld+k$, the degree of r in y. Hence r is identically 0, and $(p_l(y))^d h(x,y) = q(x,y)g(x,y)$. By using $s(y)$ as a shorthand for $(p_l(y))^d$ we get $g|h \cdot s$ and Claim 2 is proved.

The next claim is that h/g has degree d in both x and y. This proves the Lemma.

Claim 3: Assuming Claim 2, $h(x,y)/g(x,y)$ has degree at most d in x, y.

Thus far we only know that $h(x,y)/g(x,y)$ has the form $\sum_{i=0}^{10d} x^i t_i(y)$ where $t_i(y)$ is a rational polynomial whose degree in the numerator and denominator is at most $20d^3$.

But, the restriction of h/g to more than $10d$ columns is 0.5-close (by applying the reasoning of Claim 1 to columns instead of rows). By Lemma 5.6 these restrictions are degree d polynomials. It follows by interpolation that each $t_i(y)$ is a degree-d polynomial. Similarly, we can argue that the degree in x is also d. Thus Claim 3 is proved. $\square$

Proof:(Of Lemma 5.2) We show that every bivariate function with success rate $1-\beta$ (i.e. one satisfying the conditions of Lemma 5.2) satisfies the conditions of Lemma 5.3 with $\delta = 2\beta - o(1)$. This is seen as follows.

Break up all lines into equivalence classes, with a class containing only lines parallel to each other. (See Fact A.10 for a definition of *parallel*; the definition is the obvious one.) An averaging argument shows that there are two distinct classes in which the average success rate is at least $1-\beta-o(1)$. Now rotate coordinates to make these two directions the x and y directions. By construction, the best polynomials describing the rows describe f in a fraction $1-\beta-o(1)$ of points in F^2, and so do the best polynomials describing the columns. Hence the fraction of points in F^2 on which *both* the row and the column polynomial describe f is at least $1-2\beta-o(1)$. Lemma 5.3 implies now that f is $5\sqrt{2\beta}$-close to a bivariate polynomial of bidegree (d,d). We have to show that it is close to a polynomial of *total* degree d. But the fact that the success-rate is $1-\beta$ implies that the restriction of f to $1-3\beta$ fraction of lines is at least 1/3-close to a univariate polynomial of degree d. Hence Fact A.5 implies that the polynomial that f is $5\sqrt{2\beta}$-close to actually has *total* degree d.

Since $5\sqrt{2\beta} < 10\sqrt{\beta}$, we conclude that f is $10\sqrt{\beta}$-close to a bivariate polynomial of total degree d. Hence the lemma has been proved. $\square$

5.2. Correctness of the Low-degree Test

This section is devoted to proving Theorem 5.1.

Definition 5.4: A *plane in F^m* is a set of q^2 points that has a parametric representation of the form $\{\hat{u} + t_1 \cdot \hat{v} + t_2 \cdot \hat{w} : t_1, t_2 \in \mathrm{F}\}$, for some $\hat{u}, \hat{v}, \hat{w} \in \mathrm{F}^m$. Note that it is identical to the set $\{\hat{u} + t_1 \cdot c_1\hat{v} + t_2 \cdot c_2\hat{w} : t_1, t_2 \in \mathrm{F}\}$ for any $c_1, c_2 \in \mathrm{F} \setminus \{0\}$. Our convention is to fix one representation as the canonical one.

Given a bivariate polynomial h in the parameters t_1, t_2, and a function $f : \mathrm{F}^m \to \mathrm{F}$, we say that h *describes* f *at the point* $\hat{u} + a_1 \cdot \hat{v} + a_2 \cdot \hat{w}$ of the above plane iff $h(a_1, a_2) = f(\hat{u} + a_1 \cdot \hat{v} + a_2 \cdot \hat{w})$.

Definition 5.5: Let C be a plane and L_C be the set of lines in the plane. For a function $f : \mathrm{F}^m \to \mathrm{F}$ the *success rate of* f *in* C is

$$\mathrm{E}_{l \in L_C}[\mu^f(l)],$$

in other words, the average success rate of f among lines in the plane.

The general idea of the proof is that if the overall success rate of f is high, then symmetry considerations imply that f has high success rate in almost every plane in F^m. But the restriction of f to a plane is a bivariate function, so if the success rate in a plane is high, this bivariate function (by the bivariate case of the theorem, namely Lemma 5.2) is ϵ-close for some small ϵ. Hence we conclude that for almost every plane in F^m there is a bivariate polynomial (in the plane parameters t_1, t_2) that describes f almost everywhere in the plane. This implies some very strong conditions on f (namely, the statement of Lemma 5.10), which in turn imply that f itself is well-described by an m-variate polynomial.

Throughout, our symmetry-based arguments use two obvious facts. (i) Every two points in F^m together determine a unique line that contains them. (ii) Every line in F^m and every point outside it together determine a unique plane that contains them.

To state our calculations cleanly we introduce some simplification in notation. We fix the letter f to denote the function from the statement of Theorem 5.1, whose success rate is $1 - \alpha$ for some small enough α. Then we can make statements like "the line l is δ-close" instead of "the restriction of f to line l is δ-close," and "the value produced by line l at point $b \in l$," instead of "value produced by line polynomial P_l^f at the point $b \in l$," and so on.

Also, we make Lemma 5.2 more user-friendly, and use the following loose form.

Loose form of Lemma 5.2: *If the success rate of f in a plane is at least $(1 - \beta)$ for some small enough β, then the restriction of f to the plane is β-close to a bivariate polynomial.* $\square$

Note: We justify the loose form on the grounds that we are not keeping track of constants anyway. Thus the constant β in the conclusion of the loose form is not fundamentally different from the constant $10\sqrt{\beta}$ in the correct conclusion.

Now we can cleanly state an interesting property of planes with high success rate.

Definition 5.6: Let b be a point in F^m. Two lines that pass through b are said to be *coherent at* b if they produce the same value at b. Let S be a set of lines which all pass through b. The set is *c-coherent at b*, for some number $c \in [0,1]$, if it is coherent at b and in addition every line in S is c-close.

We will show (in Lemma 5.10) that a high success rate implies that for every point, most lines passing through it are coherent. As a first step, we show this fact for the bivariate case. (In the following lemma, the reader may wish to skip reading part (ii) for now and return to it later).

Lemma 5.8: *Suppose plane C has success rate $(1-\beta)$ for $\beta < 0.001$, and b is any point in C. Then the following holds.*
(i) There is a $\sqrt{\beta}$-coherent set of lines at b which contains $1-\sqrt{\beta}$ fraction of all lines in C that pass through b.
(ii) Let g be the bivariate polynomial that best describes the values of f in this plane, and let it produce the value g_b at point b. Then the value produced at b by the coherent lines in part (i) is also g_b.

Proof: If the success rate of plane C is $(1-\beta)$, then Lemma 5.2 shows that there is a unique bivariate polynomial g that describes f on $(1-\beta)$ fraction of points in $C \setminus \{b\}$. Let g_b denote the value produced by g at b.

Let S be the lines of C that pass through b. We show that the desired coherent set $S' \subseteq S$ is the one defined as

$$S' = \{l \in S : \ g \text{ describes } f \text{ on at least } 1-\sqrt{\beta} \text{ fraction of points of } l\}.$$

First we show the set is coherent at b. Let l be a line in S'. Since the restrictions of g and f agree in $1-\sqrt{\beta}$ fraction of points on l and $\sqrt{\beta} < 1/2 + d/2q$, it follows that the restriction of g to the line is also the best univariate polynomial describing l. In particular the line polynomial for the line, P_l^f, produces the value g_b at b. Since this is true for every line $l \in S'$, it follows that the lines in S' are coherent at b. Further, the definition of S' also implies that each line in S' is $\sqrt{\beta}$-close. Hence S' is $\sqrt{\beta}$-coherent at b. To complete the proof of part (i) we show that $|S'| \geq (1-\sqrt{\beta})\,|S|$.

Every point in $C \setminus \{b\}$ is connected to b by a unique line. Hence the lines in S partition $C \setminus \{b\}$. For line $l \in S$, denote by $\rho(l)$ the fraction of points on l where f and g disagree. Then $\mathrm{E}_{l \in S}[\rho(l)] = 1-\beta$. Averaging shows that $\rho(l) \geq 1-\sqrt{\beta}$ for at least $1-\sqrt{\beta}$ fraction of lines $l \in S$. Hence $|S'| \geq (1-\sqrt{\beta})\,|S|$, and part (i) is proved.

Part (ii) follows from an observation in the first para of the proof, namely that all lines in S' produce the value g_b at b. $\Box$

Note: Why is Lemma 5.8 significant? Recall the definition of success rate: it is the expectation

$$\mathrm{E}_l\,[\text{fraction of points in } l \text{ where } P_l^f \text{ describes } f\,]$$

where l is picked randomly from lines in F^m. Symmetry implies the the following expectation is also the success rate.

$$\mathrm{E}_{x \in \mathrm{F}^m}[\text{among lines intersecting } x, \text{ the fraction of those that describe } f \text{ at } x].$$

Hence high success rate implies that for "most" points x, "most" lines passing through x produce the same value at x, namely, $f(x)$. In other words, "most" points have an associated coherent set of lines and the set is large. Lemma 5.8 says that at least in the bivariate case, this is true for *all* points x and not just for "most" x. Lemma 5.10 will show that an analogous statement holds for the m-variate case as well.

First we point out the following symmetry-based fact.

Lemma 5.9: *Let p denote the success rate of f, and let b be a fixed point in F^m. Let $\mathcal{C}$ be the set of planes that contain b. Then we have*

$$E_{C \in \mathcal{C}}[\textit{success rate of } f \textit{ in } C] \geq p - o(1).$$

Proof: Recall the definition of success rate: it is the expected success rate of a line picked randomly from among all lines. But note that only a $o(1)$ fraction of all lines pass through b. Hence the expectation is almost unchanged if we pick the line randomly from among all lines that do not pass through b. Let q be this new expectation. As we just argued, $q \geq p - o(1)$.

Now we use symmetry to obtain an alternative definition of q. Every line outside b determines a unique plane containing itself and b. All planes containing b contain the same number of lines that don't contain b. So q is also the expected success rate of a line picked as follows: First pick a random plane $C \in \mathcal{C}$, and then randomly pick a line in C that does not contain b. But the new expectation is exactly the one occuring in the lemma statement, so we have proved the lemma.

$\square$

Lemma 5.10: (Main Lemma) *Let $f : F^m \to F$ have success rate $1 - \alpha$ where α is small enough, and let b be any point in F^m. Then there is a δ-coherent set at b that contains a fraction at least $(1 - 3\delta)$ of all lines passing through b, where $\delta = \sqrt[4]{\alpha}$.*

Proof: Let $b \in \mathrm{F}^m$, and L be the set of lines passing through b. Let $L_1, L_2, \ldots$ be maximal δ-coherent subsets of L satisfying (i) and (ii). Maximality implies that the L_i's are mutually disjoint. Denote by γ_i the fraction $|L_i| \,/\, |L|$, and by γ the largest of the γ_i's.

Let the term *pair* refer to a pair of lines (l_1, l_2), where l_1, l_2 are distinct lines in L. We call the pair (l_1, l_2) *agreeable* if they are both δ-close, and further, their line polynomials produce the same value at b. Notice that both the lines in an agreeable pair must come from the same subset out of $L_1, L_2, \ldots$. Hence the fraction of pairs that are agreeable is at most

$$\begin{aligned}
\sum_i \left(\frac{|L_i|}{|L|}\right)^2 &= \sum_i \gamma_i^2 \\
&\leq \gamma^2 \times \frac{1}{\gamma} \\
&= \gamma.
\end{aligned}$$

We will show that at least $1-3\delta$ of all pairs are agreeable, which implies $\gamma \geq 1-3\delta$, thus proving the Lemma.

To count the fraction of agreeable pairs, we use symmetry. Since each pair defines a unique plane containing b, and since each plane containing b contains the same number of pairs, the total number of pairs is

$$\text{Number of planes containing } b \times \text{Number of pairs per plane.}$$

Lemma 5.9 implies that the average success rate of planes passing through b is $1-\alpha-o(1) \approx 1-\delta^4$. Averaging shows that at least $1-\delta^2$ fraction of them have success rate at least $1-\delta^2$. Let C be a plane that has success rate $1-\delta^2$. Lemma 5.8 implies that of all lines in C that pass through b, at least a fraction $1-\delta$ are coherent. It follows that the fraction of pairs in C that are agreeable is at least $(1-\sqrt{\delta^2})^2$, which is at least $1-2\delta$.

We have shown that in $\geq 1-\delta^2$ fraction of planes containing b, the fraction of pairs in the plane that are agreeable is $\geq (1-2\delta)$. Hence the fraction of pairs overall that are agreeable is at least $(1-2\delta)(1-\delta^2)$, which is at least $1-3\delta$. Thus the Lemma has been proved. □

Definition 5.7: Let a function $\widehat{f} : \mathrm{F}^m \to \mathrm{F}$ be defined as

$$\widehat{f}(x) = \text{majority}_{l \ni x} \{\text{value of } P_l \text{ at } x\}.$$

(Note: Lemma 5.10 implies that this majority is an overwhelming majority.)

The following lemma shows that $\widehat{f}$ coincides with f almost everywhere.

Lemma 5.11: *Let $f : F^m \to F$ be a function with success rate $1-\alpha$, where α is a sufficiently small constant. The set of points $\left\{x \in F^m : f(x) \neq \widehat{f}(x)\right\}$ constitutes a fraction at most 2α of all points.*

Proof: Let t denote the fraction of points x such that $f(x) \neq \widehat{f}(x)$.

As noted above, one way to view the success rate is as

$$\Pr_{x,l}[P_l^f \text{ describes } f \text{ at } x],$$

where x is picked randomly from among all points and l is picked randomly from the set of lines containing x.

Let $x \in \mathrm{F}^m$ be such that $f(x) \neq \widehat{f}(x)$. Lemma 5.10 implies that function f is described at x by at most a fraction 3δ of the lines through x, where $\delta = \sqrt[4]{\alpha}$. Therefore an upper bound on the the success rate is $3\delta t + (1-t)$. Since the success rate is at least $1-\alpha$, we have $3t\delta + 1 - t \geq 1-\alpha$, which implies $t \leq \alpha/(1-3\delta) \leq 2\alpha$. □

For any line l, let $P_l^{\widehat{f}}$ denote the the univariate polynomial that best describes values of $\widehat{f}$ on l.

The following lemma shows that on every line l, the polynomial $P_l^{\widehat{f}}$ describes $\widehat{f}$ on every point of l.

Lemma 5.12: *For every line l and every point $b \in l$ the value produced by $P_l^{\widehat{f}}$ at b is $\widehat{f}(b)$.*

Proof: Let l be a line and $b \in l$ be a point.

Let y be any point on l. We say a line $l' \ni y$ is *nice for* y if l' is in y's coherent set (whose existence is proved in Lemma 5.10). In other words, l' is δ-close, and produces the value $\widehat{f}(y)$ at y. Let C be a plane containing l. We say C *is good for* y if among all lines in C that pass through y, a fraction $1 - 10^{-3}$ are nice for y.

Claim: *There is a plane C containing l that is good for b, and good also for a* 0.998 *fraction of points on l.*

First we show how to complete the proof of the lemma using the Claim. Let C be the plane mentioned in the claim. Fact A.10, part (iii) implies that line l intersects almost all the lines in C, more specifically, a fraction $1 - O(1/q)$, which is $1 - o(1)$. Since plane C is good for 0.998 fraction of points in l, the fraction of lines in C that are δ-close is $(1 - 10^{-3}) \times 0.998 \times (1 - o(1))$. Hence the success rate of C is at least $(1 - \delta) \times 0.997$, which we assume is more than 0.99. Lemma 5.2 implies there exists a bivariate polynomial h that describes f on 0.99 fraction of points of the plane. Let $z \in l$ be a point and h_z denote the value that h produces at z. Since the success rate in plane C is high, Lemma 5.8-part(ii) implies that z has an associated set of coherent lines in C, each of which produces the value h_z at z. Now suppose z is one of the points for which C is good. By definition of "good," most lines that pass through z produce the value $\widehat{f}$ at z. But we know that most lines lie in the above-mentioned coherent set for z. Hence the values $\widehat{f}(z)$ and h_z must be identical. In other words, for 0.999 fraction of points on l, including b, we have $\widehat{f}(z) = h_z$. Since $\widehat{f}$ is described by h on 0.999 fraction of points of l, the line polynomial $P_l^{\widehat{f}}$ must be the restriction of h to l. In particular, the value produced by $P_l^{\widehat{f}}$ at b must be h_b. But as noted, $h_b = \widehat{f}(b)$. Hence the Lemma is proved.

Next, we prove the Claim. We prove the existence of the desired plane probabilistically. Let $\mathcal{C}$ be the set of planes that contain l. We pick C randomly from $\mathcal{C}$. Let y be a point in l. We upperbound the probability that C is not good for y.

Define the random variable I_C to be the fraction of lines, among all lines of C that contain y, that are nice for y. Recall that every line $l' \ni y$ that is not l determines a unique plane together with l. Symmetry implies that every plane C containing l contains the same number of lines $l' \ni y$. Therefore we have

$$\mathrm{E}_{C \in \mathcal{C}}[I_C] \geq 1 - 3\delta.$$

Assume that $\sqrt{3\delta} < 10^{-3}$. Hence if plane C is not good for y then $I_C < 1 - \sqrt{3\delta}$. The Averaging Principle implies that

$$\Pr_{C \in \mathcal{C}}[\text{“}C \text{ is not good for } y\text{”}] = \Pr_{C}[I_C \leq 1 - \sqrt{3\delta}] < \sqrt{3\delta}.$$

Hence the probability that C fails the conditions of the claim is at most

$$\Pr_C[\text{“}C \text{ is not good for } b\text{”}] + \Pr_C[\text{“}C \text{ is not nice for } 0.001 \text{ fraction of points of } l\text{”}]$$
$$\leq \sqrt{3\delta} + \frac{\sqrt{3\delta}}{0.001}$$

Assuming $1000\sqrt{3\delta} + \sqrt{3\delta} < 1$, the probability that C fails the conditions of the claim is less than 1. Hence there exists a plane meeting all the conditions.

□

Now we prove Theorem 5.1, the main theorem of this chapter.

Proof:(Of Theorem 5.1) We defined a function $\widehat{f}$ in Definition 5.7. Lemma 5.12 shows that on every line the restriction of $\widehat{f}$ is described exactly by the degree-d polynomial $P_l^{\widehat{f}}$. Using Fact A.6 in the appendix, we conclude that $\widehat{f} \in \mathrm{F}_d[x_1, \ldots, x_m]$. But Lemma 5.11 shows that $\widehat{f}$ agrees with f in $1 - 2\alpha$ fraction of points. It follows that f is 2α-close. □

5.3. Discussion

The results in this chapter comprise two parts. One part, given in Section 5.2, shows that the correctness of the Low-degree Test follows from the correctness of the bivariate case of the test (in other words, from Lemma 5.2). This part is due to Rubinfeld and Sudan[RS92], although our proof is somewhat different in the way it uses symmetry arguments. The other part of the chapter, Section 5.1, shows the correctness of the bivariate case. The proof of the bivariate case is a consequence of a more general (multivariate) result in [AS92], which is stated in the next section. (The proof of the bivariate case as given here uses a little modification due to [Sud92].)

No simpler proof of the general result of [AS92] is known. But a somewhat simpler proof for the bivariate case appears in [PS94]. More importantly, that paper lowers the field size required in the lemma to $O(d)$; an improvement over $O(d^3)$ as required by our proof.

Many people have conjectured the following, although a proof still eludes us.

Conjecture 5.1: *Let $|F| = poly(d)$. If $f : F^m \to F$ is a function whose success rate is p, for $p \geq d/\sqrt{|F|}$, then f is ($\approx (1-p)$)-close to $F_d[x_1, \ldots, x_m]$.* □

If the field size $|\mathrm{F}|$ is allowed to be exponential in the degree the previous conjecture is true. We do not give the proof here.

5.3.1. History

Improvements to the Low-degree Test have accompanied most advances in the theory of probabilistically checkable proofs. The first such test, the *multilinearity test,* enabled the result MIP =NEXPTIME ([BFL91]. Subsequent tests given in [BFLS91, FGL+91] were crucial for scaling down the result for NEXPTIME to NP. An improvement in the efficiency of those tests was in turn used to prove a new characterization of NP in terms of PCP ([AS92]). Finally, the discovery of the most efficient test (the one in this section) led to the discovery of the PCP Theorem ([ALM+92]).

Actually, the above list of tests includes two kinds of low-degree tests. The first kind upper-bounds the the degree of the given function in *each* variable. The second kind – and the one in this chapter is of this kind – upper-bounds the total degree.

The tests in [BFL91, BFLS91, FGL+91, AS92] are of the first kind. (A particularly beautiful exposition of this type of test ([She91]) was never published, although a later version appears in [FSH94].) Low-degree tests of the first kind consist in estimating the success rate of the given function on *axis-parallel* lines. (A line is called axis-parallel if for some $i \in \{1, \ldots, m\}$ and $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_m \in \mathrm{F}$, it is of the form $\{(a_1, \ldots, a_{i-1}, x_i, a_{i+1}, \ldots, a_m) : x_i \in \mathrm{F}\}$.) Note, on the other hand, that the test in this chapter estimates the success rate on *all* lines. The strongest result about low-degree tests of the first kind is due to [AS92].

Theorem 5.13 ([AS92]): *Let $f : F^m \to F$ be a function. If the success rate of f on axis-parallel lines is $1 - O(1/m)$, then f is δ-close to a polynomial in $x_1, \ldots, x_m$ whose degree in each variable is at most d.*

Note that Lemma 5.3, which was used in our proof of the bivariate case, is equivalent to the special subcase $m = 2$ of the above theorem.

Low-degree tests of the second kind arose in program checking. The earliest such test appears in [BLR90], where it is called the *multilinearity test.* Subsequent tests appearing in [GLR+91, RS92] work for higher degree. In fact, these latter tests are identical to the one used in this chapter, but their analysis was not as good (it could not show that the test works when the success rate is less than $1 - O(1/d)$).

The idea of combining the work of [AS92] and [RS92] to obtain the low-degree test of this chapter is due to [ALM+92].This combination allows the test to work even when the success rate is a constant (independent of the degree). To estimate that the success rate is some high enough constant, the test needs to only pick $O(1)$ random lines, and therefore reads only $O(1)$ "chunks" of information in the provided tables. This property plays an important part in the proof of the PCP Theorem. Specifically, such a low-degree test enables queries to be "aggregated" (see Section 4.1.2), a property crucial in constructing the verifier in Theorem 3.2.

The algebraic ideas used in the proof of Lemma 5.2 were originally inspired by a Lemma of [BW], as used in [GS92].

Chapter 6

Hardness of Approximations

In Chapter 2, we proved the following consequence of the PCP Theorem: There is a constant $\epsilon > 0$ such that if MAX-3SAT can be approximated within a factor of $1 + \epsilon$, then P = NP (Corollary 2.4). Similar hardness results are now known for a host of other optimization problems, namely that if they can be approximated within some explicit factor (where the exact factor in question depends upon the problem) in polynomial time, then some well-known complexity-theoretic conjecture is false.

The reductions used to prove this body of results go via either 2 Prover 1 Round proofs for NP (a discussion on 2 Prover proofs appears in Chapter 7), or the PCP theorem. Often the best reduction for a given problem (i.e, one that proves the strongest inapproximability result for the problem) uses techniques specific to that problem, and sometimes also details of the construction of the PCP verifier.

Quite understandably, this new way of doing NP-completeness reductions is generally perceived as not very "user-friendly." Traditionally, NP-completeness reductions – which prove NP-hardness only of exact optimization – are far simpler: each one is based, in a simple way, upon one of a few canonical NP-complete problems. (The number of canonical problems in [GJ79] is six.)

In this chapter we attempt to identify problems that can serve as canonical problems for proving inapproximability results. The aim is to to derive all known inapproximability results using as few assumptions (in other words, as few canonical problems) as possible. We find that two canonical problems suffice. The first is a version of MAX-3SAT. The second is a new problem, Label Cover, defined expressly for the purpose of proving inapproximability. Note that reductions from these two cannot, for every given problem, prove the strongest possible inapproximability results for that problem. But the reductions always prove results that are in the right ball-park: for instance, the factor of approximation they prove hard is not much smaller than the best that can be proven otherwise. In some cases reductions from Label Cover prove approximation to be only *almost-NP-hard* (this term is defined in Definition 6.3), whereas direct reductions using the PCP verifier would prove it NP-hard. We could partially remedy this latter state of affairs by including a third (less natural) canonical problem in our list, but we choose not do so. The hope is that further progress in the construction of PCP -like verifiers (Conjecture 9.1 says precisely what else needs to be proved) will allow NP-hardness to be proved using Label Cover.

We first define a few terms that appear throughout this chapter.

Definition 6.1: Let Π be an optimization problem involving maximization (resp., minimization) and let $OPT(I)$ denote the value (resp., cost) of the optimum solution on input I. For a rational number $c > 1$, an algorithm is said to *c-approximate* Π if for every input I the algorithm produces a solution that has value at least $OPT(I)/c$ (resp., has cost at most $c \cdot OPT(I)$).

Note: Making c larger makes the algorithm's task easier: the set of acceptable solutions it can output gets enlarged. In many applications it makes sense to allow c to grow with the input size, since fixing c to be a constant seems to rule out polynomial time approximation algorithms. For example it makes sense to let c be $\log n$, where n is the input size. Now we define one such factor that often crops up in inapproximability results.

Definition 6.2: An approximation factor is *large* if for some fixed number ϵ in the interval $(0, \frac{1}{2})$, it is at least $2^{\log^{0.5-\epsilon} n}$, where n = input size.

Definition 6.3: A computational problem is *almost-NP-hard* if a polynomial time algorithm for it can be used to solve every NP problem in time $n^{\mathrm{poly}(\log n)}$.

Note: People believe that not only is NP $\not\subseteq$ P, but also that there are problems in NP whose solution requires more than $n^{\mathrm{poly}(\log n)}$ time. Hence proving a problem is almost-NP-hard provides a good reason to believe that it will not have polynomial-time algorithms.

Organization of the Chapter. Section 6.1 describes our canonical problems and their properties. Section 6.2 defines the gap-preserving reduction, an essential ingredient of inapproximability results. An example of a gap-preserving reduction is given using the CLIQUE problem. Section 6.3 discusses MAX-SNP, a class of optimization problems, and shows that approximating MAX-SNP-hard problems is NP-hard. Section 6.4 shows the inapproximability of a host of problems having to do with lattices, codes and linear systems. The reductions to these problems use the canonical problem Label Cover. Section 6.5 shows how to exhibit the NP-hardness of n^ϵ-approximation, where ϵ is some fixed positive constant. Section 6.6 contains a survey of known inapproximability results, and indicates how they can be proved using our canonical problems.

6.1. The Canonical Problems

Our reductions use two canonical problems, MAX-3SAT(13) and Label Cover. The latter comes in two versions, one involving maximization and the other minimization. The two versions are not really independent, since a form of *weak duality* links them (Lemma 6.1). However, we would like to keep the two versions separate, and for the obvious reason: proving the inapproximability of minimization problems (as we do in Section 6.4) requires us to have a minimization problem in our list of canonical problems.

Definition 6.4: 1. **MAX-3SAT(13)**: This is the version of MAX-3SAT in which no variable appears in more than 13 clauses. (Note: "13" has no special significance other than that it is a convenient constant.)

2. **LABEL COVER**: The input consists of: (i) A regular bipartite graph $G = (V_1, V_2, E)$[1] (ii) An integer N in unary. We think of any integer in $[1, N]$ as a *label*. (iii) For each edge $e \in E$ a partial function $\Pi_e : [1, N] \to [1, N]$.

   A *labelling* has to associate a set of labels with every vertex in $V_1 \cup V_2$. It is said to *cover* an edge $e = (u, v)$ (where $u \in V_1, v \in V_2$) if for every label a_2 assigned to v there is some label a_1 assigned to v, such that $\Pi_e(a_1) = a_2$.
   Maximization Version: Using a labelling that assigns 1 label per vertex, maximize the number of covered edges.
   Minimization Version: Using more than 1 label per vertex if necessary, cover all the edges. Minimize the *cost* of doing this, where the cost is defined as

$$\sum_{v \in V_1} (\text{number of labels assigned to } v)$$

   (that is, the total number of labels, counting multiplicities, assigned to vertices in V_1).

   Convention: To state results cleanly the value of the optimum will be stated as a ratio. In the maximization version of Label Cover, this ratio is the *fraction* of edges covered. In the minimization version this ratio is average number of labels used per vertex in V_1, that is,

$$\frac{\text{cost of labelling}}{|V_1|}.$$

   Also, the careful reader will realize that in the min. version, an optimum labelling never needs to assign more than one label to any vertex of V_2. We do not make this a part of the definition because it makes our reductions more difficult to describe.

□

The Label Cover problem was implicit in [LY94] and was defined in [ABSS93]. Although an ungainly problem at first sight, it is quite useful in reductions. We first give some examples to clarify the definition of the problem. (We currently don't see a way to simplify its definition, since all aspects of the definition seem to play a role when we do reductions.)

[1] For our purposes a bipartite graph is *regular* if for some integers d_1, d_2, every vertex on the left (resp., right) has degree d_1 (resp., d_2).

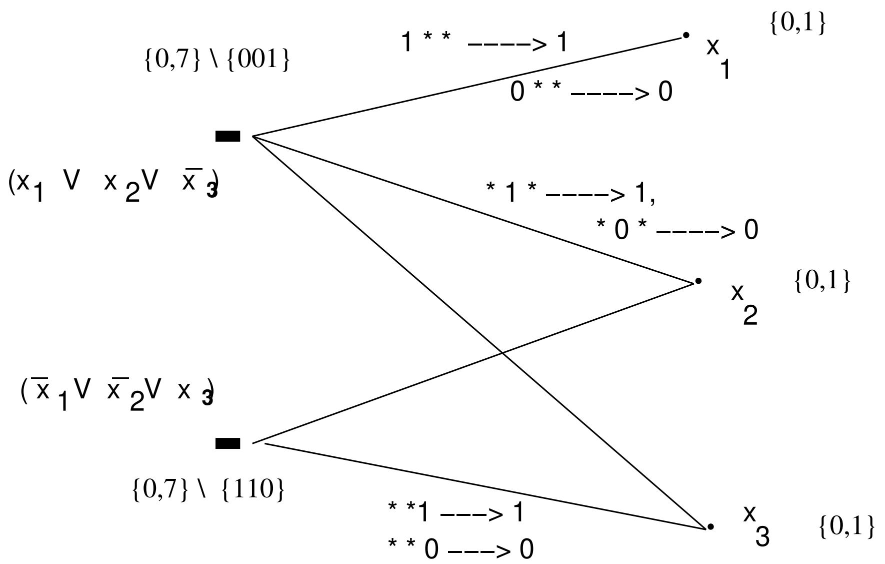

Figure 6.1: Label Cover instance for formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$. The symbols on edge e represent map Π_e.

Example 6.1: 3SAT is reducible to Label Cover (max. version). Given a 3SAT instance φ define a Label Cover instance as follows. Let V_1 have one vertex for each clause and V_2 have a vertex for every variable. let adjacency corresponds to the variable appearing in the clause, whether negated or unnegated. (For the moment we ignore the regularity condition, and leave it to the reader to fix.) The set of labels is $[0, 7]$, where the significance of the number of labels is that it is 2^3, the number of possible assignments to the 3 variables in a clause. We denote the labels in binary. For a vertex in V_1, if the corresponding clause involves variables x_i, x_j, x_k, the reader should think of a label $b_1 b_2 b_3$ as the assignment $x_i = b_1, x_j = b_2, x_k = b_3$. For a vertex in V_2, say one corresponding to variable x_i, the set of admissible labels is $\{000, 111\}$; to be thought of as assignments 0 and 1 respectively to that variable.

The edge function Π_e is described as follows. Suppose e is the edge (u, v) where $u \in V_1$ corresponds to clause $(x_1 \vee x_2 \vee \neg x_3)$, and $v \in V_2$ corresponds to variable x_1. The partial function Π_e for this edge has the domain $[0, 7] \setminus \{001\}$, in other words, the 7 assignments to x_1, x_2, x_3 that satisfy the clause. Every label of the type $x_1 = 1, x_2 = \star, x_3 = \star$ ($\star$ means "anything") is mapped under Π_e to 111 and every label of the type $x_1 = 0, x_2 = \star, x_3 = \star$ is mapped to 000.

Figure 6.1 shows the above transformation as applied to the formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$. The figure uses the shorthand "0" and "1" for 000, 111 respectively for the vertices on right.

Since each vertex is allowed only 1 label, the labels on the right hand size vertices consitute a boolean assignment to $x_1, x_2, \ldots$. The label on a left-hand vertex also constitute an assignment to the variables in the corresponding clause. The edge joining a clause-vertex and a variable-vertex is covered iff it is assigned the same value by both assignments.

Clearly, if all edges are covered then the assignment is a satisfying assignment. Conversely, if there is no satisfying assignment, some edge is left uncovered by all labellings.

Actually we can make a stronger statement, which will be useful later. Any labelling that covers a fraction ρ of edges yields an assignment that satisfies $1 - 3\rho$ fraction of the clauses.

Example 6.2: *Hitting Set* is a sub-case of Label Cover (min. version). Recall that the input to Hitting Set is a set $\mathcal{U}$, and some sets $S_1, \ldots, S_K \subseteq \mathcal{U}$. The output is the smallest subset of $\mathcal{U}$ that has a nonempty intersection with all of $S_1, \ldots, S_K$. (This problem is the dual of set-cover.)

Hitting set is a trivial reformulation of the following sub-case of Label Cover (min. version): there is just one vertex on the left (that is, $|V_1| = 1$), and for all edges e, the map Π_e has exactly one element in its image-set. (Note in particular that each vertex $v \in V_2$ has a unique edge incident to it, say e_v, and Π_{e_v} has an image-set of size 1.)

Given such an instance of Label Cover, define $\mathcal{U}$ to be $[1, N]$. Define the set $S_v \subseteq \mathcal{U}$ as $\{a_1 : \text{label } a_1 \text{ is a preimage of } \Pi_{e_v}\}$. Solving hitting set gives us the smallest set $S \subseteq [1, N]$ such that $S \cap S_v$ is nonempty for each $v \in V_2$. Then S is the solution to Label Cover, since it is the smallest set of labels that must be assigned to the lone vertex in V_1, such that all edges are covered.

A similar transformation works in the other direction: from Hitting Set to Label Cover.

The following *weak duality* (implicit in [LY94]) links the max and min versions of Label Cover. Note how our convention about stating costs/values as ratios allows a clean statement.

Lemma 6.1: *For any instance I of Label Cover, if $OPT_{\max}(I)$ and $OPT_{\min}(I)$ are, respectively, the optimum value in the max. version and the minimum cost in the min. version, then*

$$OPT_{\max}(I) \geq \frac{1}{OPT_{\min}(I)} \qquad \forall I.$$

Proof: Consider the solution of the minimization version, namely, a labelling using on an average $OPT_{\min}(I)$ labels per vertex in V_1 and covering all the edges. For any vertex $u \in V_1$ let it assign n_u labels to this vertex. Then by definition of cost

$$\sum_{u \in V_1} n_u = OPT_{\min}(I)\,|V_1|\,.$$

We randomly delete all labels for each vertex in $V_1 \cup V_2$ except one. This gives a labelling that assigns only 1 label per vertex, in other words, a candidate solution for the maximization version. We claim that the probability that any particular edge (u, v) is still covered is $1/n_u$. For, the original labelling covered (u, v); for every label a_2 it assigned to v it assigned some preimage $\Pi_e^{-1}(a_2)$ that was assigned to u. The probability that this preimage survives is $1/n_u$.

In the new (randomly constructed) labelling the expected number of edges still left covered is at least

$$\sum_{e(=(u,v))} \frac{1}{n_u}.$$

Since each vertex in V_1 has the same degree, say d, the number of edges, $|E|$, is $d\,|V_1|$, and the above expectation can be rewritten as

$$\begin{aligned}\sum_{u \in V_1} \frac{d}{n_u} &= d \sum_{u \in V_1} \frac{1}{n_u} \geq d \frac{|V_1|^2}{\sum_{u \in V_1} n_u} \\ &= d\,\frac{|V_1|^2}{OPT_{\min}(I)\,|V_1|} = \frac{|E|}{OPT_{\min}(I)}\end{aligned}$$

(The crucial fact used above is that the sum $\sum_u 1/n_u$ is minimized when all n_u are equal.)

Thus we have shown a randomized way to construct a candidate solution to the max. version, such that the expected fraction of edges covered is at least $(OPT_{\min}(I))^{-1}$. It follows that there must exist a candidate solution that covers this many edges. Hence we have proved

$$OPT_{\max}(I) \geq \frac{1}{OPT_{\min}(I)}.$$

□

Note: Lemma 6.1 uses the fact that the bipartite graph in the Label Cover instance is regular. This is the only place where we need this fact; we do not need it for our reductions.

Now we give the hardness results for our canonical problems.

Theorem 6.2: *There is a fixed positive constant ϵ for which there is polynomial time reduction from any NP problem to MAX-3SAT(13) such that YES instances map to satisfiable formulae and NO instances map to formulae in which less than $1 - \epsilon$ fraction of clauses can be satisfied.*

Proof: In Corollary 2.4 we described such a reduction for MAX-3SAT: For some fixed constant δ it satisfies the property that YES instances map to satisfiable formulae and NO instances to formulae in which no more than $1 - \delta$ fraction of the clauses can be satisfied. We show how to change instances of MAX-3SAT into instances of MAX-3SAT(13), without changing the gap of δ by much. The reduction is a slight simplification of the one in [PY91] (although their reduction yields instances of MAX-3SAT(7), and uses weaker expanders).

We need the following result about explicit constructions of expander graphs ([LPS88]): There is a procedure that, for every integer k, can construct in poly(k) time a 6-regular graph G_k on k vertices such that for every set S of size at most $k/2$, there are more than $|S|$ edges between S and its complement, $\overline{S}$.

Let the instance of 3SAT have n variables $y_1, \ldots, y_n$, and m clauses. Let m_i denote the number of clauses in which variable y_i appears. Let N denote the sum $\sum_i m_i$. Since a clause contains at most 3 variables, $N \leq 3m$.

For each variable do the following. If the variable in question is y_i, replace it with m_i new variables $y_i^1, y_i^2, \ldots,$. Use the jth new variable, y_i^j, in place of the jth occurence of y_i. Next, to ensure that the optimal assignment assigns the same value to $y_i^1, \ldots, y_i^{m_i}$, add the following $6m_i$ new clauses. For each $j, l \leq m_i$ such that (j, l) is an edge of the expander G_{m_i}, add a pair of new clauses $(y_i^l \vee \neg y_i^j)$ and $(\neg y_i^l \vee y_i^j)$. Together, this pair just says $(y_i^l \equiv y_i^j)$; an assignment satisfies the pair iff it assigns the same value to y_i^l and y_i^j.

Hence the new formula contains $6N$ new clauses and m old clauses. Each variable occurs in exactly 12 new clauses and 1 old clause. If the old formula was satisfiable, so is the new formula. Next, we show that if every assignment satisfies less than $1 - \delta$ fraction of clauses in the old formula, then no assignment satisfies less than $1 - \delta/19$ fraction of clauses in the new formula.

Consider an optimal assignment to the new formula, namely, one that satisfies the maximum number of clauses. We claim that it satisfies all new clauses. For, suppose it does not satisfy a new clause corresponding to y_i. Then it does not assign the same value to all of $y_i^1, \ldots, y_i^{m_i}$. Divide up these m_i variables into two sets S and $\overline{S}$ according to the value they were assigned. One of these sets has size at most $m_i/2$; say it is S. In the expander G_{m_i}, consider the set of $|S|$ vertices corresponding to vertices in S. Expansion implies there are at least $|S| + 1$ edges leaving this set. Each such edge yields an unsatisfied new clause. Hence by flipping the value of the variables in S, we can satisfy at least $1 + |S|$ clauses that weren't satisfied before, and possibly stop satisfying the (at most $|S|$) old clauses that contain these variables. The net gain is still at least 1. Therefore our assumption that the assignment we started with is optimal is contradicted.

We conclude that $\forall i \in \{1, \ldots, n\}$, the optimal assignment assigns identical values to the different copies $y_i^1, \ldots, y_i^{m_i}$ of y_i. Now suppose no assignment could satisfy more than $(1-\delta)m$ clauses in the original formula. Then in the new formula no assignment can satisfy more than $6N + (1 - \delta)m$ clauses. Since $N \leq 3m$, we see that the fraction of unsatisfied clauses is at least $\frac{\delta}{18m+m} = \delta/19$. Hence the theorem has been proved for any value of ϵ that is less than $\delta/19$.

□

The following corollaries are immediate.

Corollary 6.3: *There is an $\epsilon > 0$ such that finding $(1 + \epsilon)$-approximations to MAX-3SAT(13) is NP-hard.* □

Corollary 6.4: *Finding $(1+\epsilon/3)$ approximations to Label Cover (max. version) is NP-hard, where ϵ is the same as in Corollary 6.3.*

Proof: Use the reduction from 3SAT to Label Cover in Example 6.1. As observed there, any labelling that covers a fraction $1-p$ of the edges for some p yields an assignment that satisfies $1-3p$ of the clauses.

□

Approximating Label Cover even quite weakly is also hard.

Theorem 6.5: *Let ρ be any large factor (as defined in Definition 6.2). There is a $n^{\mathrm{poly}(\log n)}$-time reduction from any NP problem to Label Cover (max. version) such that YES instances map to instances in which the optimum value is 1, and NO instances map to instances in which the optimum value is less than $1/\rho$.*

Proof: Proved in Chapter 7. □

Hence we have the following corollary.

Corollary 6.6: *Approximating Label Cover (max. version) within any large factor is almost-NP-hard.* □

The following theorem gives the hardness of approximating the min. version of label-cover.

Theorem 6.7: *Approximating the minimization version of Label Cover within any large factor is almost-NP-hard. More specifically, for every large factor ρ, there is a $n^{poly(\log n)}$-time reduction from NP to Label Cover (min. version) that maps YES instances to instances in which optimum cost is 1, and NO instances to instances in which the optimum cost is more than ρ.*

Proof: Consider the instances of Label Cover arising out of the reduction in Theorem 6.5, by using the same value of ρ. The optimum value of the max. version is either 1 or less than $1/\rho$. When the optimum cost is 1, there is a labelling that uses 1 label per vertex and and covers all the edges, hence the optimum cost of the min. version is 1. When the optimum value in the max. version is less than $1/\rho$, Lemma 6.1 implies the optimum cost of the min. version is at least ρ. □

6.2. Gap-Preserving Reductions

In Theorems 6.3 and 6.6 we proved that approximating our canonical problems is hard. To extend these inapproximability result to other problems, we need to do reductions carefully, so that they are gap-preserving.

Definition 6.5: Let Π and Π' be two maximization problems and $\rho, \rho' > 1$. A *gap-preserving reduction with parameters* $(c, \rho), (c', \rho')$ from Π to Π' is polynomial-time algorithm f. For each instance I of Π, algorithm f produces an instance $I' = f(I)$ of Π'. The optima of I and I', say $OPT(I)$ and $OPT(I')$ respectively, satisfy the following property.

$$\text{if } OPT(I) \geq c \text{ then } OPT(I') \geq c' \text{ and if } OPT(I) \leq c/\rho \text{ then } OPT(I') \leq c'/\rho'. \tag{6.1}$$

□

Example 6.3: For a 3CNF formula φ, define $\mathrm{OPT}(\varphi)$ as the maximum fraction of clauses that can be satisfied by an assignment. Recall the expander-based reduction from MAX-3SAT to MAX-3SAT(13) in Theorem 6.2. It maps formulae in which $OPT = 1$ to formulae in which $OPT = 1$. Further, it maps formulae in which $\mathrm{OPT} \leq 1 - \delta$ to formulae in which $\mathrm{OPT} \leq 1 - \delta/19$.

Hence the reduction is gap-preserving with parameters $(1, (1-\delta)^{-1}), (1, (1-\delta/19)^{-1})$, for every positive fraction δ.

Comments on Definition 6.5: *(1)* Suppose there is a polynomial time reduction from NP to Π such that YES instances are mapped to instances of Π of cost $\geq c$ and NO instances to instances of cost $\leq c/\rho$. Then the reduction of Lemma 6.5 (if it exists) implies that finding ρ'-approximations to Π' is NP-hard. *(2)* Like most known reductions ours will also map solutions to solutions in an obvious way. For instance, given a solution to I' of cost $> c'$, a solution to I of cost $\geq c$ can be produced in polynomial time. But we keep this aspect out of the definition for simplicity. *(3)* The above definition can be modified in an obvious way when one (or both) of the optimization problems involve minimization. *(4)* A gap-preserving reduction, since its "niceness" (namely, equation 6.1) holds only on a partial domain, is a weaker notion than the *L-reduction* introduced in ([PY91]), where "niceness" has to be maintained on *all* instances of Π. An L-reduction, coupled with an approximation algorithm for Π', yields an approximation algorithm for Π. The previous statement is false for a gap-preserving reduction. On the other hand, for exhibiting merely the hardness of approximation, it suffices (and is usually easier) to find gap-preserving reductions. (Indeed, for some of the problems we will cover later, L-reductions are not known to exist.)

Often our gap preserving reductions work with values of the factors ρ, ρ' that are functions of the input length. The reduction in the following result is in this vein. Let CLIQUE be the problem of finding the largest clique (i.e., a vertex-induced subgraph that is a complete graph) in the given graph.

Theorem 6.8: *For every $\epsilon > 0$, finding $2^{\log^{0.5-\epsilon} n}$-approximations to CLIQUE is almost-NP-hard.*

Proof: We give a gap-preserving reduction from Label Cover (max. version) to CLIQUE such that the optimum in the CLIQUE instance is exactly the optimum value of the Label Cover instance. In other words, for every $c, \rho > 0$, the reduction satisfies the parameters $(c, \rho), (c, \rho)$. Since Label Cover is hard to approximate within any large factor (Lemma 6.5), it follows that so is CLIQUE.

Let (E, V_1, V_2, N, Π) be an instance of Label Cover.

For an edge e and labels a_1, a_2, let a *labelling scenario* (or scenario for short) be a triple (e, a_1, a_2) such that $\Pi_e(a_1) = a_2$. If $e = (u, v)$, we think of the scenario as assigning label a_1 to vertex u and label a_2 to v. (Clearly, any labelling that assigned labels this way would cover e.) Two scenarios are *inconsistent with each other* if the two edges involved share a vertex, and the scenarios assign different labels to this shared vertex. (To give an example, the scenarios $((u, x), a_1, a)$ and $((u, y), a_1', b)$ are inconsistent regardless of what x, a, y, b are, since one scenario assigns label a_1 to u while the other assigns a_1'.)

Now we construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ using the given instance of Label Cover.

The set of vertices in $\mathcal{G}$ is the set of labelling scenarios. Two different scenarios are adjacent in $\mathcal{G}$ iff they are not inconsistent.

Claim: Size of largest clique in $\mathcal{G}$ = Number of edges covered by optimum label-cover.

The proof is in two parts.

($\geq$) For any labelling of $V_1 \cup V_2$ that covers K edges, take the set of scenarios occuring at the covered edges. All the scenarios in this set are consistent with one another, so they form a clique in $\mathcal{G}$. The clique has size K.

($\leq$) For any clique $S \subseteq \mathcal{V}$ of size K construct as follows a labelling that assigns labels to a subset of vertices, and covers K edges. (Clearly, such a partial labelling can be extended to a complete labelling that covers at least K edges.) Notice that no two scenarios in a clique can be inconsistent with one another. Hence for any vertex in $V_1 \cup V_2$, no two scenarios present in the clique assign different label to that vertex. Now assign to each vertex any label (if one exists) that gets assigned to it by a scenario in the clique. This defines a (partial) labelling that covers every edge e which appears in a scenario in S. The number of such edges is K.

$\square$

Note: The graph $\mathcal{G}$ produced by this reduction has special structure: it is a union of $|E|$ independent sets (for any edge $e = (u, v)$, the two distinct vertices (e, a_1, a_2) and (e, a_1', a_2') must have either $a_1 \neq a_1'$, or $a_2 \neq a_2'$ and so are not connected in $\mathcal{G}$). Further, the size of the largest clique is either $|E|$ or $|E|/\rho$. This property of the graph $\mathcal{G}$ is useful in doing a further reduction to Chromatic Number (see Section 6.6).

6.3. MAX-SNP

NP-hard optimization problems exhibit a vast range of behaviors when it comes to approximation. Papadimitriou and Yannakakis ([PY91]) identified a large sub-class of them that exhibit the same behavior. The authors defined a class of optimization problems, MAX-SNP, as well as a notion of completeness for this class. Roughly speaking, a MAX-SNP-complete problem behaves just like MAX-3SAT in terms of approximability. This made MAX-3SAT a plausible candidate problem to prove hard to approximate, and in particular motivated the discovery of the PCP theorem.

MAX-SNP contains constraint-satisfaction problems, where the constraints are local. More formally, the constraints are definable using a quantifier-free propositional formula. The goal is to satisfy as many constraints as possible.

Definition 6.6: A maximization problem is in MAX-SNP if given an instance I we can in polynomial time write a structure G and a quantifier-free formula $\phi(G, S, \overline{x})$ with the following properties. Formula ϕ involves the relations in G, a relation symbol S (not part of G) and a vector of variables $\overline{x}$, where each variable in $\overline{x}$ takes values over the universe of G.

The value of the optimum solution on instance I, $\mathrm{OPT}(I)$, is given by

$$\mathrm{OPT}(I) = \max_S |\{\overline{x} : \phi(G, S, \overline{x}) = \text{ TRUE}\}|.$$

Note: The above definition is inspired by Fagin's model-theoretic characterization of NP ([Fag74]), and an explanation is in order for those unfamiliar with model theory. G consists of a sequence of relations of fixed arity, over some universe $\mathcal{U}$. If $\mathcal{U} = n$, then G implicitly defines an "input" of size $O(n^k)$ where k is the largest arity of a relation in G. The role of S is that of a "nondeterministic guess."

Example 6.4: Let MAX-CUT be the problem of partitioning the vertex-set of an undirected graph into two parts such that the number of edges crossing the partition is maximised. Here's how to see it is in MAX-SNP. The universe is the vertex-set of the graph. Let G consist of E, a binary relation whose interpretation is "adjacency." Let S be a unary relation (interpreted as one side of the cut), and $\phi(E, S, (u, v)) = E(u, v) \wedge (S(u) \neq S(v))$.

Both MAX-3SAT and MAX-3SAT(13) are also in MAX-SNP. For every MAX-SNP problem, there is some constant $c \geq 1$ such that the problem can be c-approximated in polynomial time ([PY91]). The smallest such value of c for MAX-CUT is approximately 1.13 ([GW94]), for example.

There is a notion of *completeness* in the class MAX-SNP. According to the original definition, a MAX-SNP problem is complete for the class if every MAX-SNP problem can be reduced to it using an L-reduction. We are interested in hardness of approximation.

For us it will suffice to consider a problem *MAX-SNP-hard* if every MAX-SNP problem has a gap-preserving reduction to it with parameters ρ, ρ' some absolute constants greater than 1 (See the note following Definition 6.5 to see why the original definition is more restrictive.) Examples of MAX-SNP-hard problems include MAX-CUT, MAX-3SAT(13), and many others. (See Section 6.6 for a partial list.) From Theorem 6.3 the following is immediate.

Corollary 6.9: *For every MAX-SNP-hard problem, there exists some $c > 1$ such that finding c-approximations to it is NP-hard.* $\square$

6.4. Problems on Lattices, Codes, Linear Systems

This section contains inapproximability results for a large set of NP-hard functions. All the results involve gap-preserving reductions from Label Cover. The functions considered include well-known minimum distance problems for integral lattices and linear codes as well as the problem of finding a largest feasible subsystem of a system of linear equations (or inequalities) over $\mathbf{Q}$. In this section, n denotes the length of the input and m the dimension of the lattice, code etc. under consideration.

6.4.1. The Problems

An *integral lattice* $\mathcal{L}(b_1, \ldots, b_m)$ in $\mathbf{R}^k$, generated by the basis $\{b_1, \ldots, b_m\}$ is the set of all linear combinations $\sum_{i=1}^{m} \alpha_i b_i$, where the $\{b_i\}$ is a set of independent vectors in $\mathbf{Z}^k$ and $\alpha_i \in \mathbf{Z}$.

Definition 6.7: (Shortest Vector Problem in ℓ_p norm, SV_p)*: Given a basis $\{b_1, \ldots, b_m\}$, find the shortest non-zero vector (in ℓ_p norm) in $\mathcal{L}(b_1, \ldots, b_m)$.*

Definition 6.8: *(*Nearest Vector Problem in ℓ_p norm, NV$_p$*): Given a basis $\{b_1, \ldots, b_m\}$, and a point $b_0 \in \mathbf{Q}^k$, where $b_0 \neq 0$, find the nearest vector (in ℓ_p norm) in $\mathcal{L}(b_1, \ldots, b_m)$ to b_0.*

Next we define three other problems all of which in one way or another involve distances of vectors.

Definition 6.9: Nearest Codeword: INPUT : An $m \times k$ matrix A over $GF(2)$ and a vector $y \in GF(2)^k$.
OUTPUT : A vector $x \in GF(2)^m$ minimizing the Hamming distance between xA and y.

Max-Satisfy INPUT : A system of k equations in m variables over $\mathbf{Q}$.
OUTPUT : (Size of the) largest subset of equations that is a feasible system.

Min-Unsatisfy INPUT : A system of k equations in m variables over $\mathbf{Q}$.
OUTPUT : (Size of) The smallest set of equations whose removal makes the system feasible.

The next problem is a well-known one in *learning theory:* learning a halfspace in the presence of malicious errors. The problem arises in the context of training a perceptron, a learning model first studied by Minsky and Papert [MP68]. Rather than describing the learning problem in the usual PAC setting([Val84]), we merely present the underlying combinatorial problem.

The input to the learner consists of a set of k points in $\mathbf{R}^m$, each labelled with $+$ or $-$. (These should be considered as positive and negative examples of a concept.) The learner's output is a hyperplane, $a \cdot x = b$ $(a, x \in \mathbf{R}^m, b \in \mathbf{R})$. The hypothesis is said to *correctly classify* a point marked $+$ (resp. $-$) if that point, say y satisfies $a \cdot y > b$ ($a \cdot y < b$, resp.). Otherwise it is said to *misclassify* the point.

Finding a hypothesis that minimizes the number of misclassfications is the *open hemispheres* problem, which is NP-hard [GJ79]. Define the *error* of the algorithm as the number of misclassifications by its hypothesis, and the *noise* of the sample as the error of the best possible algorithm. Let the *failure ratio* of the algorithm be the ratio of the error to noise.

Definition 6.10: Failure Ratio. Input: A set of k points in $\mathbf{R}^m$, each labelled with $+$ or $-$.
Output: A hypothesis that makes the ratio of error to noise 1.

Note that to c-approximate Failure Ratio means to output a hypothesis whose failure ratio is at most c.

6.4.2. The Results

We prove the following results.

Theorem 6.10: *1. Approximating each of NV_p, Nearest Codeword, Min-Unsatisfy, and Failure Ratio (i) within any constant factor $c \geq 1$ is NP-hard; (ii) within any large-factor is almost-NP-hard.*

2. Approximating SV_∞ within any large factor is almost-NP-hard.

We note that our reductions use only vectors/systems with all entries $0, \pm 1$. Hence it follows that approximation in those sub-cases is equally hard. Part 2 of Theorem 6.10 is proved in Section 6.4.6. Part 1 follows easily from Lemma 6.11, in conjunction with the hardness result for Label Cover (Theorem 6.7). Lemma 6.11 is proved in Section 6.4.5 (although an important gadget used there is described earlier in Section 6.4.4).

Lemma 6.11: *For each of the problems in part 1 of Theorem 6.10, and every factor ρ there is some cost c and some polynomial time gap-preserving reduction from Label Cover (min. version) to the given problem which has parameters $(1,\rho),(c,\rho')$, where $\rho'=\rho$ for all the reductions except for NV_p, where it is $\sqrt[p]{\rho}$.*

Note the values of the parameters, more specifically, the fact that the reduction maps Label Cover instances with optimum cost at most 1 to instances of the other problem with optimum cost at most c. By definition, the cost of the min version cannot be less than 1. So what the gap-preserving reduction actually ensures is that Label Cover instances with optimum cost *exactly* 1 are mapped to instances with cost at least c. This is an example of a reduction for which we do not know whether an L-reduction (in the sense of [PY91]) exists. However, the gap-preserving reduction as stated above is still good enough for proving inapproximability in conjunction with the reduction stated in Theorem 6.7, since that other reduction (quite conveniently, it seems) did map YES instances to Label Cover instances with optimum cost exactly 1.

For Max-Satisfy we will prove a stronger result in Section 6.5.1.

Theorem 6.12: *There is a positive constant ϵ such that finding n^{ϵ}-approximations to Max-Satisfy is NP-hard.*

Better Results For NEAREST-CODEWORD and NV_p for all $p \geq 1$, we we can prove almost-NP-hardness up to a factor $2^{\log^{1-\epsilon} n}$ instead of $2^{\log^{0.5-\epsilon} n}$. These results appear in [ABSS93]. Also, in our reductions the number of variables, dimensions, input size etc. are polynomially related, so n could be any of these.

Previous or Independent Work Bruck and Naor ([BN90]) have shown the hardness of approximating the NEAREST-CODEWORD problem to within some $1+\epsilon$ factor. Amaldi and Kann ([AK93]) have independently obtained results similar to ours for MAX-SATISFY and MIN-UNSATISFY.

6.4.3. Significance of the Results

We discuss the significance of the problems we defined.

Lattice Problems. The SV problem is particularly important because even relatively poor polynomial-time approximate solutions to it (within c^m, [LLL82]) have been used in a host of applications, including integer programming, solving low-density subset-sum problems and breaking knapsack-based codes [LO85], simultaneous diophantine approximation and factoring polynomials over the rationals [LLL82], and strongly polynomial-time algorithms in combinatorial optimization [FT85]. For details and more applications, especially to classical problems in the "geometry of numbers", see the surveys by Lovász [Lov86] or Kannan [Kan87].

Lovász's celebrated lattice transformation algorithm [LLL82] runs in polynomial time and approximates SV_p ($p \geq 1$) within c^m. A modification of this algorithm [Bab86] yields the same for NV_p. Schnorr modified the Lovász algorithm and obtained, for every $\epsilon > 0$, approximations within $O(2^{\epsilon m})$ in polynomial time for these problems [Sch85].

On the other hand, Van Emde Boas showed that NV_p is NP-hard for all $p \geq 1$ ([vEB81]; see [Kan87] for a simpler proof). Lagarias showed that the shortest vector problem is NP-hard under the ℓ_∞ (i.e. max) norm. But it is still an open problem whether SV_p is NP-hard for any other p, and specifically for $p = 2$.

While we do not solve this open problem, we obtain hardness results for the approximate solutions of the known NP-hard cases.

We mention that improving the large factors in either the NV_2 or the SV_∞ result to $\sqrt{m}$ (m = dimension) would prove hardness of SV_2, a long standing open question. The reason is that approximating either SV_∞ or NV_2 to within a factor $\sqrt{m}$ is reducible to SV_2. To see this for SV_∞, notice that the solutions in SV_∞ and SV_2 are always within a factor $\sqrt{m}$ of each other. For NV_2 the implication follows from Kannan's result [Kan87] that approximating NV_2 within a factor $\Omega(\sqrt{d})$ is reducible to SV_2.

We also note that approximating NV_2 within any factor greater than $m^{1.5}$ is unlikely to be NP-complete, since Lagarias et al. [LLS90] have shown that this problem lies in $\mathrm{NP} \cap \text{co-NP}$.

Problems on Linear Systems. Note that a solution to MAX-SATISFY is exactly the complement of a solution to MIN-UNSATISFY, and therefore the two problems have the same complexity. (Indeed, it is known that both are NP-hard; this is implicit e.g. in [JP78]). However, the same need not be true for approximate solutions. For instance, vertex-cover and independent-set are another "complementary" pair of problems, and seem to differ greatly in how well they can be approximated in polynomial time. (Vertex cover can be approximated within a factor 2, and independent-set is NP-hard up to a factor n^c for some $c > 0$ [FGL$^+$91, AS92, ALM$^+$92].)

We find it interesting that large factor approximation is hard for our two complementary problems. We do not know of any other complementary pair with the same behavior.

We know of no good approximation algorithms for any of these problems. Kannan has shown us a simple polynomial time algorithm that uses Helly's theorem to approximate MIN-UNSATISFY within a factor of $m + 1$.

Failure Ratio. That the failure ratio is hard to approximate has often been conjectured in learning theory [HS92]. We know of no good approximation algorithms. A failure ratio $\leq m$ can be achieved by Kannan's idea, mentioned above.

6.4.4. A Set of Vectors

In this section we use Label Cover instances to define a set of vectors that will be used by all the reductions.

Let (V_1, V_2, E, Π, N) be an instance of Label Cover (min version). Think of a labelling as a pair of functions $(\mathcal{P}_1, \mathcal{P}_2)$, where the set of labels assigned to the vertex $v_1 \in V_1$, is $\mathcal{P}_1(v_1)$, and the set of labels assigned to the vertex $v_2 \in V_2$ is $\mathcal{P}_2(v_2)$. Let us call a labelling that covers every edge a *total cover.*

First simplify the structure of the edge functions Π_e. For a vertex $v_1 \in V_1$ prune as follows the domains of the edge functions of each edge containing it. Restrict the set of labels that can be assigned to v_1 to contain only labels $a_1 \in [1, N]$ such that for every edge e incident to v_1, there is a label a_2 such that $\Pi_e(a_1) = a_2$. In other words, prune Π_e so that the domain contains only those labels a_1 that can be used to cover all edges incident to v_1. Call such a label *valid* for v_1 and call (v_1, a_1) a valid pair.

Allowing only valid labels to be assigned to vertices does not hurt us. The reduction, being gap-preserving with parameters $(1, \rho)$), can assume that the total cover uses either exactly 1 label per vertex, or at least ρ labels per vertex on average. In the former case, each label used must be valid. And in the latter case, restricting the set of possible (vertex, label) pairs to be valid can only increase the minimum cost of a total cover.

The set of vectors contains a vector $V_{[v_2,a_2]}$ for each $v_2 \in V_2$ and $a_2 \in [1, N]$, and a vector $V_{[v_2,a_2]}$ for each *valid pair* (v_1, a_1), where $v_1 \in V_1$ and $a_1 \in [1, N]$. Note that any linear combination of these vectors implicitly defines a labelling, where label a is assigned to vertex v iff $V_{[v,a]}$ has a nonzero coefficient in the combination.

Definition 6.11: Let $x = \sum c_{[v_i,a_i]} \cdot V_{[v_i,a_i]}$ be a linear combination of the vectors in the set, where the coefficients $c_{[v_i,a_i]}$ are integers. The *labelling defined by the vector* x, denoted $(\mathcal{P}_1^x, \mathcal{P}_2^x)$, is the one that assigns to $v_1 \in V_1$ the set of labels $\mathcal{P}_1^x(v_1) = \{a_1 \mid c_{[v_1,a_1]} \neq 0\}$. The set of labels $\mathcal{P}_2^x(v_2)$ assigned to $v_2 \in V_2$ is defined similarly.

Define the vectors in the set as follows. Each has $|E|(1 + N)$ coordinates, $1 + N$ consecutive coordinates for each edge $e \in E$. Call the coordinates corresponding to e in a vector as its *e-projection.* (See Figure 6.2.)

For $j = 1, 2, \ldots, N$, let u_j be a vector with $1 + N$ coordinates, in which the j'th entry is 1 and all the other entries are 0. With some abuse of notation we'll associate the vector u_{a_2} with the label $a_2 \in [1, N]$. Let $\vec{0}$ and $\vec{1}$ be the all-zero vector and all-one vectors, respectively.

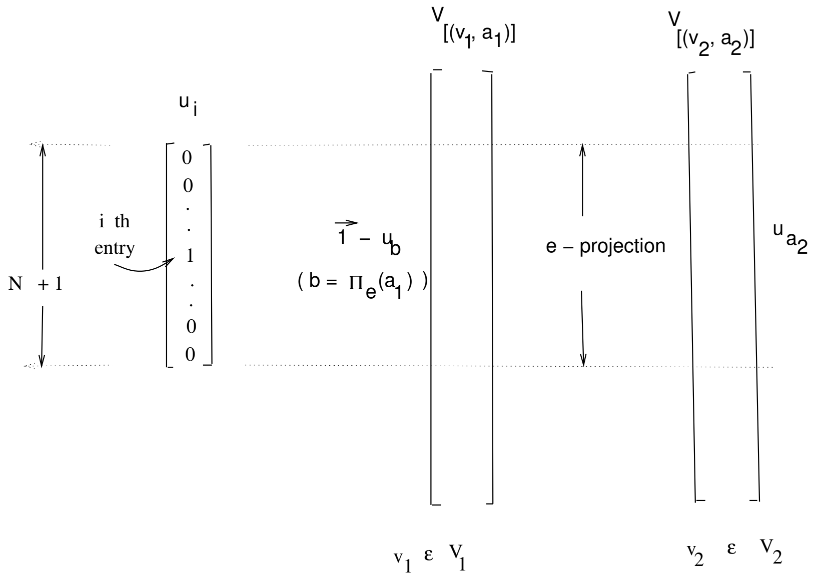

Figure 6.2: Figure showing the e-projections of vectors $V_{[(v_1,a_1)]}$ and $V_{[(v_2,a_2)]}$ in the vector set, where $e = (v_1, v_2)$.

For $v_2 \in V_2, a_2 \in [1, N]$, let the e-projection of the vector $V_{[v_2,a_2]}$ be u_{a_2} if e is incident to v_2; and $\vec{0}$ otherwise.

For each valid pair v_1, a_1, let the e-projection of the vector $V_{[v_1,a_1]}$ be $\vec{1} - u_{\Pi_e(a_1)}$ if e is incident to v_1; and $\vec{0}$ otherwise.

Note that the e-projections of the vectors form a multi-set comprised of exactly one copy of the vector u_{a_2} for each label a_2, zero or more copies of the vector $\vec{1} - u_{a_2}$, and multiple copies of $\vec{0}$. The following lemma says that a linear combination of vectors in the multiset is $\vec{1}$ iff for some label a_2 both the vectors u_{a_2} and $\vec{1} - u_{a_2}$ appear in it.

Lemma 6.13: *Suppose some integer linear combination of the vectors* $\{u_{a_2} | a_2 \in [1, N]\} \cup \left\{\vec{1} - u_{a_2} | a_2 \in [1, N]\right\}$ *is* $\vec{1}$. *Then there is some* $a_2 \in [1, N]$ *such that the coefficients of both* u_{a_2} *and* $\vec{1} - u_{a_2}$ *in the linear combination are nonzero.*

Proof: The vectors $\{u_1, \ldots, u_N\}$ are linearly independent and do not span $\vec{1}$. Therefore if

$$\sum_{a_2} (c_{a_2} \cdot u_{a_2} + d_{a_2} \cdot (\vec{1} - u_{a_2})) = \vec{1}$$

then $c_{a_2} = d_{a_2}$ for all a_2. Furthermore, there exists an a_2 for which these are not zero. □

Corollary 6.14: *If x is a nontrivial linear combination of the vectors $\{V_{[v_i,a_i]}\}$ and $x = \vec{1}$, then $(\mathcal{P}_1^x, \mathcal{P}_2^x)$ is a total cover.*

Proof: For any edge $e = (u, v)$, the e-projections of the vectors $\{V_{[v_i,a_2]}\}$ form a system described in the hypothesis of Lemma 6.13. Note that the e-projections of the type u_{a_2} belong to the vector of $V_{[v,a_2]}$, and the e-projection $(\vec{1} - \vec{u_{a_2}})$ belong to a vector $V_{[u,a_1]}$ such that $\Pi_e(a_1) = a_2$. The fact that the linear combination x assigns nonzero coefficients to both $\vec{u_{a_2}}$ and $(\vec{1} - \vec{u_{a_2}})$, for some a_2, implies that $(\mathcal{P}_1^x, \mathcal{P}_2^x)$ assigns some label a_2 to v and a a_1 to u such that $\Pi_e(a_1) = a_2$. Since this happens for every e, labelling $(\mathcal{P}_1^x, \mathcal{P}_2^x)$ covers all edges and is a total-cover. □

6.4.5. Reductions to $\mathbf{NV}_1$ and others

Now we prove Lemma 6.11.

Proof: (Lemma 6.11) First, we show the reduction to the Nearest Lattice Vector problem with the ℓ_1 norm; results for the other problems will follow easily. The main idea in the reduction is to use the set of vectors $\left\{V_{[v_i,a_i]}\right\}$ from Section 6.4.4 as part of the lattice definition. The fixed point is chosen in such a way that all vectors that are near to it involve an integer linear combination of the set $\left\{V_{[v_i,a_i]}\right\}$ that is $\vec{1}$. Thus vectors near to the fixed point are forced to define a total cover (as seen in Corollary 6.14).

The basis vectors have $|E| \cdot (1 + N) + |V_1| \cdot N$ coordinates: $|V_1| \cdot N$ more than vectors in the previous section.

Let L be the integer $|E| \cdot (1 + N)$. The fixed point, W_0, will have an L in each of the first $|E|(1 + N)$ dimensions and 0 elsewhere.

The basis of the lattice consists of the following vectors: for every vector in the above set $\left\{V_{[v_i,a_i]}\right\}$, there is a basis vector $W_{[v_i,a_i]}$. In the first $|E| \cdot (1 + N)$ coordinates, the vector $W_{[v_i,b_i]}$ equals $L \cdot V_{[v_i,b_i]}$. We think of the last $|V_1| \cdot N$ coordinates as being identified one-to-one with a valid pair (v_1, a_1). Then the coordinate identified with (v_1, a_1) contains 1 in $W_{[v_1,a_1]}$ and 0 in all other vectors.

Since corresponding to every basis vector there is a unique coordinate in which this vector is 1 but no other vector is, the following claim is immediate.

Claim: Let $x = \sum c_{[v_i,a_i]} \cdot W_{[v_i,a_i]}$ be a vector in the lattice. Then $|| - W_0 + x||_1 \geq |V_1| \cdot \text{cost of } (\mathcal{P}_1^x, \mathcal{P}_2^x)$ (where cost is being measured as usual as a ratio).

Now let OPT be the minimum cost of a total cover. We show that every vector x in the lattice satisfies $|| - W_0 + x||_1 \geq \min\{L, |V_1| \cdot \mathrm{OPT}\}$. Notice that each entry of $W_0 - x$ in the first $|E|(1 + N)$ dimensions is a sum of integer multiples of L. If it isn't 0, its magnitude is $\geq L$, and so $|| - W_0 + x||_1 \geq L$. On the other hand, if all those entries are 0 then, by Corollary 6.14, $(\mathcal{P}_1^x, \mathcal{P}_2^x)$ is a total-cover, and so by the above claim $|| - W_0 + x||_1 \geq |V_1| \cdot \mathrm{OPT}$.

Finally, if there is a total-cover $(\mathcal{P}_1, \mathcal{P}_2)$ of cost 1, then the following vector has length $|V_1|$.
$x = -W_0 + \sum_{v_1 \in V_1} W_{[v_1, \mathcal{P}_1(v_1)]} + \sum_{v_2 \in V_2} W_{[v_2, \mathcal{P}_2(v_2)]}$. □

Now we show the hardness of the other problems. Let m be the number of basis vectors, and U be an integer such that the number of coordinates is $m + U$.

NV_1 with 0/1 vectors. Replace each of the last U coordinates by a set of L new coordinates. If a vector had an L in the original coordinate, it has a 1 in each of the new L coordinates, and 0 otherwise.

Other Finite Norms. Changing the norm from ℓ_1 to ℓ_p changes the gap from c to $\sqrt[p]{c}$, hence the result claimed for ℓ_p norms also follows.

l_∞ Norm. See [ABSS93] for details

Nearest-Codeword. View the vectors $b'_1, \ldots, b'_m$ obtained from the reduction to "NV_1 with 0/1 vectors" as generators of a binary code. Let the received message be b'_0. Then the minimum distance of b'_0 to a codeword is exactly K in one case and $c \cdot K$ in the other.

Min-Unsatisfy. Consider the instance $b'_0, b'_1, \ldots, b'_m$ again. Each vector has $m + L \cdot U$ coordinates. This instance implicitly defines the following system of $m + L \cdot U$ equations in m variables :

$$-b'_0 + \sum \alpha_i \cdot b'_i = \vec{0},$$

where the α_i's are the variables and $\vec{0}$ is the vector whose all coordinates are 0. The structure of the problem assures us that if we satisfy the first $L \cdot U$ equations, then the non-zero variables must yield a set-cover (and then each nonzero variable gives rise to an unsatisfied equation among the last m ones). Thus the minimal number of equations that must be removed in order to yield a satisfiable system is K in one case, and $\geq c \cdot K$ in the other.

Learning Halfspaces. Notice that minimizing the failure ratio involves solving the following problem: Given a system of strict linear inequalities, find the smallest subset of inequalities whose removal from the system makes it feasible. Now we take the system of equations in the MIN-UNSATISFY reduction and replace each equation by two inequalities in the obvious way. This does not give a system of *strict* inequalities, however, it does give a gap in the number of inequations that must be removed in order to make the system feasible. The inequalities can be made strict (for this special case) by introducing a new variable δ and changing each inequation $... + .. \geq 0$ to $... + .. + \delta > 0$, at the same time introducing L identical new inequations $\delta < 1/L$. It is now easily seen that any solution to the largest feasible subsystem must have $\delta < 1/L$, which in turn forces the variables to be 0/1. □

6.4.6. Hardness of Approximating SV_∞

Proving the correctness of our reduction to SV_∞ involves delving into the geometric structure of Label Cover instances produced with a specific proof system, namely, the one due to Feige-Lovász proof-system [FL92]). We do not know whether a gap-preserving reduction exists from Label-Cover.

For the reduction to SV_∞ we'll need to prove the hardness of a related (and not very natural) covering problem.

Definition 6.12: Let (V_1, V_2, E, Π, N) be an instance of Label Cover (min. version). Let $(\mathcal{P}_1, \mathcal{P}_2)$ be a labelling and $e = (v_1, v_2)$ be an edge. The edge is *untouched* by the labelling if $\mathcal{P}_1(v_1)$ and $\mathcal{P}_2(v_2)$ are empty sets. It is *cancelled* if $\mathcal{P}_2(v_2)$ is empty, $\mathcal{P}_1(v_1)$ is not empty, and for every $b_1 \in \mathcal{P}_1(v_1)$ there is an $b_1' \in \mathcal{P}_1(q_1)$ such that both (e, b_1, a_2) and (e, b_1', a_2) are in Π for some $a_2 \in A_2$.

Definition 6.13: A labelling $(\mathcal{P}_1, \mathcal{P}_2)$ is a *pseudo-cover* if it assigns a label at least one vertex, and every edge is either untouched, cancelled or covered by it.

Note that in a pseudo-cover the only case not allowed is that for some edge (v_1, v_2), the set of labels $\mathcal{P}_1(v_1)$ is empty but $\mathcal{P}_2(v_2)$ is not.

Definition 6.14: The ℓ_∞ *cost* of a labelling $(\mathcal{P}_1, \mathcal{P}_2)$ is

$$\max \left\{ |\mathcal{P}_1(v_1)| : v_1 \in V_1 \right\}.$$

One of our main theorems is that approximating the minimum ℓ_∞ cost of a pseudo-cover is hard.

Lemma 6.15: *For every large factor ρ, there is a $n^{poly(\log n)}$-time reduction from any NP language to instances of Label-Cover, such that YES instances map to instances which have a pseudo-cover with ℓ_∞ cost 1, and NO instances map to instances where every pseudo-cover has has ℓ_∞ cost at most ρ.*

We indicate in Chapter 7 how this lemma is proved. Now we show the hardness of approximating SV_∞.

Theorem 6.16: *For any large factor ρ, approximating SV_∞ within a factor $\sqrt{\rho}$ is almost-NP-hard.*

To prove the theorem we again use the vectors from Section 6.4.4.

Lemma 6.17: *Let $\{V_{[v_i,a_i]}\}$ be the set of vectors defined for the Label Cover instances of Lemma 6.15, constructed as in Section 6.4.4. If x is a nontrivial linear combination of the vectors and $x = \vec{0}$, then $(\mathcal{P}_1^x, \mathcal{P}_2^x)$ is a pseudo- cover.*

Proof: For any edge $e = (u, v)$, the e-projections of the vectors $\{V_{[v_i,a_2]}\}$ form a system described in the hypothesis of Lemma 6.13. Note that the e-projections of the type u_{a_2} belong to the vector of $V_{[v,a_2]}$, and the e-projection $(\vec{1} - \vec{u_{a_2}})$ belong to a vector $V_{[u,a_1]}$ such that $\Pi_e(a_1) = a_2$. Since the set of vectors u_{a_2} are linearly independent, it follows that for every linear combination of these vectors that is $\vec{0}$, for every label a_2, either both $\vec{u_{a_2}}$ and $(\vec{1} - \vec{u_{a_2}})$ have coefficient 0, or both have a non-zero coeeficient. Hence in terms of the labelling $(\mathcal{P}_1^x, \mathcal{P}_2^x)$ defined by this combination, if $\mathcal{P}_2^x$ assigns some label a_2 to v then $\mathcal{P}_1^x$ must assign a label a_1 to u such that $\Pi_e(a_1) = a_2$. Since this happens for every e, labelling $(\mathcal{P}_1^x, \mathcal{P}_2^x)$ pseudo-cover. □

The reduction uses an $\ell \times \ell$ Hadamard matrix i.e. a (± 1) matrix such that $H_\ell^t H_\ell = \ell I_\ell$. ($H_\ell$ exists e.g. when ℓ is a power of 2, cf. [Bol86, p.74]).

Lemma 6.18: *Let $z \in \mathbf{Z}^\ell$. If z has at least k nonzero entries then $||H_\ell z||_\infty \geq \sqrt{k}$.*

Proof: The columns of $\frac{1}{\sqrt{\ell}} H_\ell$ form an orthonormal basis. Hence $||\frac{1}{\sqrt{\ell}} H_\ell z||_2 = ||z||_2 \geq \sqrt{k}$. □

Proof:(*of Theorem 6.16*) We use the set of vectors from Lemma 6.17, and extend them (by adding new coordinates) to get the basis set for our lattice.

Let L be the integer $|E||A_2|$. The vectors in the basis have $|E| \cdot (1 + N) + |V_1| \cdot N$ coordinates each, that is, $|V_1| \cdot N$ more than the vectors of Lemma 6.17. For each vector $V_{[v_i,a_i]}$ of that other set, the basis contains a vector $W_{[v_i,b_i]}$. The basis also contains an additional vector, W_0, that has L in each of the first $|E| \cdot (1 + N)$ coordinates and 0 elsewhere.

As in the NV_1 reduction, $W_{[v_i,a_i]}$ will equal $L \cdot V_{[v_i,a_i]}$ in the first $|E| \cdot (1+N)$ coordinates. The remaining $|V_1| \cdot N$ coordinates will be viewed as blocks of N coordinates, each associated with a $v_1 \in V_1$. We refer to entries in the block associated with v_1 as v_1-*projection* of the vector.

We may assume there exists a Hadamard matrix H_ℓ for $\ell = N$. With each label $a_1 \in [1, N]$ we identify a unique column vector of H_l, denoted h_{a_1}. Then the v_1-projection of $W_{[v,a_1]}$ is h_{a_1} if $v = v_1$ and $\vec{0}$ if $v \neq v_1$.

Let OPT be the minimum ℓ_∞-cost of a pseudo-cover.

Claim: For any vector x in the lattice $||x||_\infty \geq \sqrt{OPT}$.

Proof: The entry in any of the first $|E|(1 + |A_2|)$ coordinates is a sum of integer multiples of L, so if it is not 0, its magnitude is $\geq L$, and hence $\geq$ OPT. So all these entries must be

0. But then by Lemma 6.14, we conclude that the labelling defined by x is a pseudo-cover, and must therefore assign $\geq$ OPT labels to some $v_1 \in V_1$. But then $||x||_\infty \geq \sqrt{\text{OPT}}$ by Lemma 6.18.□

Finally, if OPT $= 1$ and $(\mathcal{P}_1, \mathcal{P}_2)$ achieves it, then the following vector has ℓ_∞ norm 1.

$$-W_0 + \sum_{v_1 \in V_1} W_{[v_1, \mathcal{P}_1(v_1)]} + \sum_{v_2 \in V_2} W_{[v_2, \mathcal{P}_2(v_2)]}.$$

□

6.5. Proving n^ϵ-approximations NP-hard

In this section we give some idea of how to prove the NP-hardness of n^ϵ-approximations. Original proofs of these results were more difficult, and we present a simplification due to [AFWZ93]. All the results could also be proved in a weaker form using reductions from Label Cover, but that would prove only almost-NP-hardness of large factors (although see the open problems in Chapter 9; specifically Conjecture 9.1).

Often problems in this class have a *self-improvement property.* We illustrate this property with the example of clique.

Definition 6.15: Given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, their *product*[2] $G_1 \times G_2$ is the graph whose vertex-set is the set $V_1 \times V_2$, and edge-set is

$$\{((u_1, v_1), (u_2, v_2)) : (u_1, u_2) \in E_1 \text{ and } (v_1, v_2) \in E_2\}.$$

Example 6.5: Let $\omega(G)$ be the size of the largest clique in a graph. It is easily checked that $\omega(G_1 \times G_2) = \omega(G_1)\omega(G_2)$.

Now suppose a reduction exists from 3SAT to clique, such that the graph G produced by the reduction has clique number either l, or $(1-\epsilon)l$, depending on whether or not the 3SAT formula was satisfiable. In other words, $(1-\epsilon)^{-1}$-approximation of clique number is NP-hard. We claim that as a consequence, any constant-factor approximation is NP-hard. Consider G^k, the kth power of this graph. Then $\omega(G^k)$ is either l^k or $(1-\epsilon)^k l^k$; by increasing k enough, the gap in clique numbers, $(1-\epsilon)^{-k}$, can be made arbitrarily large. This is what we mean by self-improvement.

Note however that G^k has size n^k, so k must remain $O(1)$ if the above construction has to work in polynomial time.

[2]There exist other ways of defining graph-products as well.

The rapid increase in problem size when using self-improvement may seem hard to avoid. Surprisingly, the following combinatorial object (specifically, as constructed in Theorem 6.19) often allows us to do just that.

Definition 6.16: Let n be an integer. A (k, n, α) *booster* is a collection $\mathcal{S}$ of subsets of $\{1, \ldots, n\}$, each of size k. For every subset $A \subseteq \{1, \ldots, n\}$, the sets in the collection that are subsets of A constitute a fraction between $(\rho - \alpha)^k$ and $(\rho + \alpha)^k$ of all sets in $\mathcal{S}$, where $\rho = \frac{|A|}{n}$.

Convention: When $\rho < \alpha$, the quantity $(\rho - \alpha)^k$ should be considered to be 0.

Example 6.6: The set of all subsets of $\{1, \ldots, n\}$ of size k is a booster with $\alpha \approx 0$. This is because for any $A \subseteq \{1, \ldots, n\}$, $|A| = \rho n$, the fraction of sets contained in A is $\binom{\rho n}{k} / \binom{n}{k}$, which is $\approx \rho^k$. The problem with this booster is that its size is $\binom{n}{k} = O(n^k)$, hence k must be $O(1)$ for any use in polynomial time reductions.

The following theorem appears in [AFWZ93]. Its proof uses explicit constructions of expander graphs ([GG81]).

Theorem 6.19: *For any $k = O(\log n)$ and $\alpha > 0$ an (n, k, α) booster of size poly(n) can be constructed in poly(n) time.* $\square$

Let G be a graph on n vertices. Using any (n, k, α) booster – for any k, α – we can define a *booster product* of G. This is a graph whose vertices are the sets of the booster $\mathcal{S}$, and there is an edge between sets $S_i, S_j \in \mathcal{S}$ iff $\forall u, v \in S_i \cup S_j$ either $u = v$ or $\{u, v\}$ is an edge in G.

Lemma 6.20: *For any graph G, and any (k, n, α) booster, the clique number of the booster product of G lies between $(\omega(G) - \alpha)^k |\mathcal{S}|$ and $(\omega(G) + \alpha)^k |\mathcal{S}|$.*

Proof: Let $A \subseteq \{1, \ldots, n\}$ be a clique of size $\omega(G)$ in graph G. Then the number of sets from $\mathcal{S}$ that are subsets of A is between $[(\omega(G) - \alpha)^k |\mathcal{S}|, (\omega(G) + \alpha)^k |\mathcal{S}|]$. Clearly, all such sets form a clique in the booster product.

Conversely, given the largest clique B in the booster product, let A be the union of all sets in the clique. Then A is a clique in G, and hence must have size at most $\omega(G)$. The booster property implies that the size of B is as claimed. $\square$

Theorem 6.21: *Approximating Clique within a factor n^ϵ for some $\epsilon > 0$ is NP-hard.*

Before proving Theorem 6.21 (which is due to [ALM+92]), we prove a weaker result about a related problem, Vertex Cover. Let $VC_{\min}(G)$ denote the size of the minimum vertex cover in graph G.

Theorem 6.22: *There exist fixed constants c, ϵ and a polynomial time reduction from a SAT instance φ to a graph with n vertices, $m = O(n)$ edges and degree 4, such that if φ is satisfiable then $VC_{min} = cn$, and if φ is not satisfiable then $VC_{min} \geq (1+\epsilon)cn$.*

Proof: Consider the transformation from MAX-3SAT(13) to Vertex Cover in [GJ79], p.54. The degree of any vertex is no more than the maximum number of clauses that a variable appears in, in this case 13. So in particular, the number of edges is linear in the number of vertices. Further, the reduction is gap-preserving (in the sense of Definition 6.5) with $\rho, \rho' = \theta(1)$. Hence by combining this reduction with the hardness result for MAX-3SAT(13) (Theorem 6.2) we get the desired result. $\Box$

Corollary 6.23: *The statement in Theorem 6.22 holds for Clique as well, except the graph produced by the reduction may not have linear size.*

Proof: In general for any graph G we have (see [GJ79] again)

$$\omega(\overline{G}) = n - VC_{\min}(G),$$

where $\overline{G}$ is the complement graph of G, that is, a graph with the same vertex set as G but with edge-set $\{(u,v) : (u,v) \notin G\}$.

Let G be the graph of Theorem 6.22. Then $\omega(\overline{G})$ is either $(1-c)n$ or $(1-(1+\epsilon)c)n$. Thus there is a gap of $\Theta(n)$ between the clique numbers in the two cases. This proves the theorem. (Of course, the number of edges in $\overline{G}$ is not linear in n; it is $\theta(n^2)$.) $\Box$

Now we prove that n^ϵ-approximation to clique is NP-hard.

Proof:(*Of Theorem 6.21*) Take the graph G from Corollary 6.23, which we know has clique number either $\geq cn$ or $\leq c(1-\beta)n$ for some fixed c, β.

Now construct a $(n, \log n, \alpha)$ booster, $\mathcal{S}$, using Theorem 6.19, by choosing $\alpha < c\beta/100$ (say). Construct the booster product of G. Lemma 6.20 says the clique number is either $\approx c^{\log n} |\mathcal{S}|$ or $\approx ((1-\beta)c)^{\log n} |\mathcal{S}|$. Hence the gap is now n^γ for some $\gamma > 0$, and further, $|\mathcal{S}| = \text{poly}(n)$, so this gap is $|\mathcal{S}|^\epsilon$ for some $\epsilon > 0$. $\Box$

For a history of the various results on the Clique problem, see the note at the end of the section.

We must emphasize that for some problems, including chromatic number ([LY94, LY93]) the known hardness results for a factor n^ϵ do not use a self-improvement property. Instead a direct reduction is given from instances of Clique obtained in Theorem 6.21, and the following property (ensured by a careful reduction and booster construction) is crucial: the graph obtained from the clique reduction is r-partite for some r, and furthermore, in one of the cases, the clique number is exactly r. It is open whether a reduction exists in the absence of this property.

Finally we note that reductions from Label-Cover are easier to describe than the above reductions (e.g., see the Clique result in Theorem 6.8), and also have the above-mentioned r-partiteness property. In other words, we can prove the hardness of approximating Chromatic Number in a somewhat easier fashion when we reduce from Label Cover. This is one of the reasons why we prefer Label-Cover as a canonical problem. However, reductions from Label-Cover prove almost-NP-hardness instead of NP-hardness, and there is room for improvement there.

An open problem. Are all self-improvable problems NP-hard to approximate within a factor n^{ϵ} for some $\epsilon > 0$? A possible exception might be the Longest Path problem ([KMR93]), for which we do not know a booster construction analogous to the one given above for Clique.

6.5.1. MAX-SATISFY

This section proves the hardness of n^{δ}-approximations to MAX-SATISFY (Definition 6.9).

Proof:(of Theorem 6.12) We first show the hardness of $(1+\epsilon)$-approximations, and then use self-improvement to show hardness for n^{δ}-approximation, just as for the Clique problem.

We will reduce from the vertex cover instances of Theorem 6.22 to a system of $N = n+3m$ linear equations, of which at most $2m+n-VC_{min}$ can be simultaneously satisfiable. This implies a gap of $\Theta(N)$ in the optimum in the two cases, since $m = O(n)$.

For each vertex i there is a variable x_i and an equation $x_i = 0$. For each edge, $\{i,j\}$, there are 3 equations:

$$x_i + x_j = 1, \quad x_i = 1 \quad x_j = 1.$$

Notice that at most 2 of the 3 equations for each edge can be satisfied simultaneously. Further, to satisfy 2 of these equations, x_i and x_j must take on values from $\{0,1\}$ and at least one must take the value 1.

We claim that the maximum number of equations are satisfied when all the x_i's are 0/1. Suppose, under some assignment, x_i is not 0/1. Then note that the following resetting of variables strictly increases the number of satisfied equations: If $x_i < \frac{1}{2}$ then set $x_i = 0$, and if $x_i \geq \frac{1}{2}$ then set $x_i = 1$. Hence the optimum setting is a 0/1 setting.

Now notice that under any optimal assignment, the set of vertices $\{i : x_i = 1\}$ constitutes a vertex cover. For, if not, then there must be be an edge $\{i,j\}$ such that both x_i and x_j are 0. Thus all three equations associated with this edge are unsatisfied. Resetting x_i to 1 will satisfy 2 equations associated with this edge, and violate one equation, $x_i = 0$, which was previously satisfied. Thus there is a a net gain of 1 equation, which contradicts the assumption that the original assignment was optimal. It follows that the optimum assignment satisfies $2m + n - VC_{min}$ equations.

Self-Improvement. Suppose we have (as above) N equations, in which the answer to MAX-SATISFY is either OPT or OPT$\cdot(1-\delta)$ for some $\delta > 0$. Let the equations be written as $p_1 = 0, p_2 = 0, \ldots, p_N = 0$. Let k, T be integers (to be specified later). The new set of equations contains, for every k-tuple of old equations, $p_{i_1}, \ldots, p_{i_k}$, a set of T equations $\sum_{j=1}^{k} p_{i_j} y^j = 0$, where $y = 1, 2, \ldots, T$. Thus the number of equations is $\binom{N}{k} \cdot T$.

Using the fact that a polynomial of degree k has at most $k+1$ roots, it is easily seen that then the number of equations that can be satisfied in the two cases is either $\geq \binom{OPT}{k} \cdot T$ or $\leq \binom{N}{k} \cdot k + \binom{OPT(1-\delta)}{k} \cdot T$. By choosing $T \geq N^{k+1}$, we see that the gap between the optima in the two cases is approximately $(1-\delta)^k$.

Now it should be clear that instead of using the trivial booster, namely, the set of all subsets of size k, we can use the booster of Theorem 6.19. Write down T equations for every subset of k equations that form a set in the booster. Use $k = \log N$, $\alpha < \delta c/100$. Thus the NP-hardness of N^ϵ-approximation follows. $\Box$

6.6. Other Inapproximability Results: A Survey

This section contains a brief survey of other known inapproximability results. All can be proved using reductions from MAX-3SAT(13) and Label Cover. Original proofs for many used direct reductions from 2 Prover 1 Round proof systems (see Chapter 7 for a definition). The Label Cover problem extracts out the the combinatorial structure of these proof systems, so we can modify all those reductions easily to use only Label Cover. We do not go into further details here.

The hardness results using Label Cover are somewhat peculiar. They don't just use the fact that Label Cover is hard to approximate, but the following stronger result: For Label Cover (max. version), it is hard to distinguish between instances in which the optimum is exactly 1 and those in which the optimum is at most $1/\rho$, where ρ is a large factor (see Theorem 6.5). For an example of such a peculiar hardness result see the comment after Theorem 6.7.

Problems seem to divide into four natural classes based upon the best inapproximability result we can prove for them.

Class I. This class contains problems for which $(1+\epsilon)$-approximation, for some fixed $\epsilon > 0$, is NP-hard. (The value of ϵ may depend upon the problem.) All the problems known to be in this class are MAX-SNP-hard (as defined in Section 6.3). Consequently Corollary 6.9 implies the above inapproximability result for them. The following is a partial list of MAX-SNP-hard problems : MAX-SAT, MAX-2SAT(13), Independent Set, Vertex Cover, Max-Cut (all in [PY91]), Metric TSP ([PY93b]), Steiner Tree ([BP89]), Shortest Superstring ([BJL$^+$91]), Multiway Cuts([DJP$^+$92]), and 3-D Matching ([Kan92]). Many more continue to be found. Since [ALM$^+$92], there have been improvements in the value of the constant ϵ for which the above results are known to hold. Currently the constant is of the order of 10^{-2} for most problems, and about 0.02 for MAX-3SAT ([BGLR93, BS94]).

Class II. This class contains problems for which $\Omega(\log n)$-approximation is hard. Currently it contains only Set Cover and related problems like Dominating Set. A reduction in [LY94] shows that if there is a polynomial-time algorithm that computes $\omega(\log n)$-approximations for these then all NP problems can be solved in $n^{O(\log\log n)}$ time. The reduction can be trivially modified to work with Label Cover. (It is also known that any constant factor approximation is NP-hard [BGLR93]; this result can also be proved using Label Cover.)

Class III. This class contains problems for which ρ-approximation is almost-NP-hard, where ρ is a large factor (large factors are defined in Definition 6.2). These problems may be further divided into two subclasses, based upon how inapproximability is proved for them.

Subclass IIIa. This contains problems for which inapproximability results are based upon Label Cover. Some of these problems are Nearest Lattice Vector, Nearest Codeword, Min-Unsatisfy, Learning Halfspaces in presence of error (all in [ABSS93] and in Section 6.4), Quadratic Programming ([FL92, BR93]), and an entire family of problems called MAX-π-Subgraph([LY93]). A problem is in MAX-π-subgraph if it involves computing, for some fixed non-trivial graph property π that is closed under vertex deletion, the largest induced subgraph of the given graph that has property π. A recent result of Raz (see Section 7.2) improves the inapproximability result for these somewhat: they are hard upto a factor $2^{\log^{1-\delta} n}$ for some $\delta > 0$.

Subclass IIIb. This contains self-improvable problems for which we do not know of a booster-type construction analogous to the one given for the Clique problem (Section 6.5). The way to prove these results is to first prove that $(1+\epsilon)$-approximation is NP-hard (using reductions from MAX-3SAT(13)), and then use self-improvement (see Example 6.5) to get a hardness result for a factor $2^{\log^{1-\delta} n}$ for some $\delta > 0$. This set of problems includes Longest Path ([KMR93]) and the Nearest Codeword problem ([ABSS93], although for the latter a more direct reduction is given in Section 6.4). The Label Cover Problem is also in this class, although with a weaker notion of self-improvement (see Section 7.2).

Class IV. This contains problems for which n^{ϵ}-approximation is NP-hard, for some $\epsilon > 0$. The class includes Clique and Independent Set ([ALM+92], see Section 6.5), Chromatic Number ([LY94]), Max-Planar-Subgraph, the problem of computing the largest induced planar subgraph of a graph, ([LY93]), Max-Set-Packing, and constrained versions of the 21 problems in Karp's original paper (the last two results are in [Zuc93]). All these results are provable using MAX-3SAT(13).

The lone problem that does not fit into the above classification is the Shortest Vector Problem using the ℓ_∞ norm. The reduction to it outlined in Section 6.4.6 uses in an intimate way the structure of the proof-system in [FL92], specifically, the fact that the protocol involves a higher dimensional geometry of lines and points.

Finally, we mention a recent result by Ran Raz ([Raz94], see Chapter 7 for an introduction) that allows the hardness of Label Cover to be proved using just the hardness result for MAX-3SAT(13). However, since we need the peculiar hardness result for Label Cover that was described at the beginning of this section, we need to use something stronger than just the fact that MAX-3SAT(13) is hard to approximate. We need Theorem 6.2: For some fixed $\epsilon > 0$, it is NP-hard to distinguish between instances of MAX-3SAT(13) that are satisfiable, and instances in which every assignment satisfies less than a fraction $1 - \epsilon$ of the clauses.

In particular, Raz's result implies that all known hardness results (except the above-mentioned version of the Shortest Lattice Vector) can now be derived from Theorem 6.2. But his proof is very complicated, so it seems prudent to just retain Label Cover as a canonical problem in our list.

6.7. Historical Notes/Further Reading

The lure of proving better inapproximability results for Clique has motivated many developments in the PCP area. The first hardness result for Clique was obtained in [FGL+91]. NP-hardness (of constant-factor approximation) was proved in [AS92]. Further, as observed first in [Zuc91], the constant factor hardness result can be improved to larger factors by using a pseudo-random graph-product. The result of [AFWZ93] stated in Section 6.5 is the cleanest statement of such a construction. The NP-hardness of n^{ϵ}-approximation is due to [ALM+92], although with a different reduction (namely, the one due to [FGL+91], plus the idea of [Zuc93]). The connection between MAX-SNP and Clique (albeit with a randomized booster construction) was first discovered in [BS92].

A result in [Zuc93] shows the hardness of approximating the k times iterated log of the clique number, for any constant k.

Free bits. The constant ϵ in the Clique result has seen many improvements [ALM+92, BGLR93, FK94b, BS94]. The latest improvements center around the concept of *free bits* ([FK94b]). This is a new parameter associated with the PCP verifier that is upperbounded by the number of query bits, but is often (e.g., in the verifier we constructed) much smaller. Improvements in this parameter lead directly to an improvement in the value of ϵ in the Clique result. As a result of many optimizations on this parameter, Bellare et al. have recently shown that if there is a polynomial-time algorithm that $\sqrt[3]{n}$-approximates Clique, then every NP language has a subexponential randomized algorithm. (A related result says $\sqrt[6]{n}$-approximation to Clique is NP-hard.) This result also implies that Chromatic Number is hard to approximate upto a factor of $\sqrt[5]{n}$ ([Für94]).

Finally, we must mention older inapproximability results that are not based upon PCP. These include a result on the hardness of approximating the unrestricted Travelling Salesman Problem ([SG76]), and a result about an entire class of maximal subgraph problems ([Yan79]).

Further reading. A recent unpublished survey by the author and Carsten Lund ([AL95]) provides a more comprehensive treatment of results on the hardness of approximations than the one given here. For a listing of optimization problems according to their approximation properties, consult [CK94].

Chapter 7

PCP Verifiers that make 2 queries

The verifiers referred to in the title of this chapter are better known as 2 Prover 1 Round interactive proof systems. These systems constitute an alternative probabilistic setting– somewhat different from the PCP setting – in which NP has been studied. Our reason for renaming them is that, as noted by many researchers, their definition ([BGKW88, FRS88]) specializes that of PCP.

We need them to prove the hardness of approximating Label Cover, one of our two canonical problems in Chapter 6. For this reason our definition is geared to a careful study of Label Cover, and is therefore less general than the definition of 2 Prover 1 Round systems.

Definition 7.1: A *restricted PCP verifier* inherits all the properties of the verifier in the definition of PCP (see the description before Definition 2.4). In addition it has the following restrictions.

1. **Uses a certain alphabet.** The proof has to be a string in Σ^*, where Σ is the verifier's alphabet (Σ depends upon the input size).
2. **Expects two tables in the proof.** The proof has to consist of two tables, T_1 and T_2. A certain length (depending upon the input size) is prescribed for each table.
3. **Makes two randomly-distributed queries.** The verifier reads the symbol in one location each in both T_1 and T_2. That location in T_1 (resp., T_2) is chosen uniformly at random from among all locations in T_1 (resp., T_2).
4. **Expects T_2 to confirm what T_1 says.** Suppose we fix the verifier's random string, and thus also the locations it queries in T_1 and T_2. For every choice of symbol $a_1 \in \Sigma$, there is at most one symbol $a_2 \in \Sigma$ such that the verifier accepts upon reading a_1 in T_1 and a_2 in T_2.

Note: Condition 3, some kind of a *regularity condition*, does not require that the queries to tables T_1 and T_2 come from independent distributions. For example, if both tables have

t entries, the following way of generating queries is legal, since both queries are uniformly distributed in $[1, t]$.

> Pick i randomly from $[1, t]$. Query location i in T_1 and location $(i+1) \bmod t$ in T_2.

Definition 7.2: For integer valued functions r, s, p, a language L is in $\mathrm{RPCP}(r(n), s(n), p(n))$ if there is a restricted verifier which on inputs of size n uses $O(r(n))$ random bits, an alphabet of size $2^{O(s(n))}$ (that is, every symbol can be represented by $O(s(n))$ bits); and satisfies:

- If input $x \in L$, there is a proof Π such that the verifier accepts for every choice of random string (i.e., with probability 1).
- If input $x \notin L$, the verifier accepts no proofs with probability more than $2^{-p(n)}$.

Example 7.1: We give an example of an RPCP verifier to clarify the definition. Hopefully, it will also motivate the connection to Label Cover, since the verifier's program is intimately related to the Label Cover instance constructed in Example 6.1.

Let L be a language in NP. We give a RPCP verifier for L that uses $O(\log n)$ random bits, examines 3 bits in T_1, and 1 bit in T_2. There is a fixed positive constant ϵ (independent of the input size) such that if an input is not in L then the verifier rejects with probability at least $1 - \epsilon$.

Given any input x, the verifier reduces it to an instance φ of MAX-3SAT(13) by using the reduction of Theorem 6.2. Assume every variable of φ appears in exactly 4 clauses, and every clause has exactly 3 literals (this can be arranged).

The verifier expects the tables to be structured as follows. Table T_1 has to contain, for each clause in φ, a sequence of 3 bits representing an assignment to the variables of this clause, and T_2 to contain, for each variable in φ, a bit representing the assignment to this variable. The verifier picks a clause uniformly at random from among all clauses of φ, and a variable uniformly at random out of the 3 variables appearing in it. It accepts iff the 3 bits given for this clause in T_1 satisfy the clause, and if this assignment is consistent with the assignment to this variable in T_1.

If $x \notin L$, every assignment in T_1 fails to satisfy a fraction ϵ of the clauses. Hence the verifier rejects with probability at least $\epsilon/3$. The fact that the the queries are uniformly distributed follows from the special structure of φ. Also, the verifier satisfies Condition 4 of the definition, since it accepts iff the value assigned by T_2 to a variable confirms the value assigned by T_1 to that variable.

The following theorem represents the best construction about restricted PCP verifiers. (A recent result by Raz, described later, improves it.)

Theorem 7.1 ([FL92]): *For all integers $k \geq 2$,*

$$NP \subseteq RPCP(\log^{2k+2} n, \log^{k+2} n, \log^k n).$$

□

7.1. Hardness of Approximating Label Cover

Label Cover captures exactly the underlying combinatorial structure of an RPCP verifier: a verifier yields, once the input has been fixed, an instance of Label Cover. Vertices of the bipartite graph (V_1, V_2, E) represent locations in the tables T_1 and T_2. The set of edges corresponds to the set of possible random strings the verifier can use. The labels correspond to the verifier's alphabet. The edge functions represent the verifier's decisions upon reading pairs of labels. The regularity condition on the verifier's queries ensures that the graph thus produced is regular, as required by definition of Label Cover. Now we describe this construction.

Fix an input x. This fixes the length of the tables T_1 and T_2 in the proof, say they are n_1, n_2 respectively; the size of the verifier's alphabet, say N; and the number of choices for the random string, say R. We identify symbols in the alphabet with numbers in $[1, N]$, and the set of possible random strings of the verifier with with numbers in $[1, R]$.

Condition 3 on the query to T_1 being uniformly distributed implies that for each location q_1 in T_1,

$$|\{r : r \in [1, R] \text{ and } r \text{ causes } q_1 \text{ to be queried}\}| = \frac{R}{n_1}. \tag{7.1}$$

Similarly, each location in T_2 is queried by R/n_2 random strings.

Note that fixing the verifier's random string to r fixes the locations it queries in T_1 and T_2, say q_1 and q_2 respectively. If V accepts using random seed r and reading a_1 in location q_1 and a_2 in q_2, we denote this by $V(r, a_1, a_2) = 1$. Condition 4 in Definition 7.1 implies that

$$\forall r \in [1, R], a_1 \in [1, N] \text{ there is a unique } a_2 \in [1, N] : V(r, a_1, a_2) = 1. \tag{7.2}$$

Now we construct the instance of label-cover. The graph (V_1, V_2, E) has $|V_1| = n_1, |V_2| = n_2$. Vertices in $V_1 \cup V_2$ are identified 1-to-1 with locations in T_1 and T_2. Let the set of edges E be defined as

$$\{(u, v) : u \in V_1, v \in V_2 \text{ and } \exists\ r \in [1, R] \text{ using which } V \text{ queries these locations}\}.$$

Thus we can identify E and R in a one-to-one fashion, and $|E| = R$. The condition in Equation 7.1 implies that the graph (V_1, V_2, E) is regular.

Let the set of labels be $[1, N]$. For $e = r = (u, v)$, define the partial function $\Pi_e : [1, N] \to [1, N]$ as

$$\Pi_e(a_1) = a_2 \text{ iff } V(r, a_1, a_2) = 1.$$

The condition in Equation 7.2 implies that Π_e is well-defined as a partial function.

The construction can be performed in time which is at most polynomial in the running time of the verifier, and $R + N$, and produces instances of size $O(RN)$.

Lemma 7.2: *The optimum value of the max. version of label-cover on the above instance is exactly*

$$|E| \times \max\{\Pr[V \text{ accepts } \pi \text{ on input } x] : \pi \text{ a proof-string}\}.$$

Proof: In this lemma, "labelling" refers to an assignment of 1 label per vertex in $V_1 \cup V_2$. The set of labellings is in 1-to-1 correspondence with the set of possible ways to construct the tables T_1, T_2. For any edge $e = (u, v) \in E$, if $r \in [1, R]$ is the corresponding random string, then labels a_1, a_2 assigned to u, v respectively cover e iff $V(r, a_1, a_2) = 1$. Thus the set of edges covered by a labelling is exactly the set of random strings for which the verifier accepts the corresponding tables T_1, T_2. $\Box$

Theorem 6.5) is a simple corollary to Theorem 7.1 and Lemma 7.2.

Proof: *(Of Theorem 6.5)* Let $L \in$ NP. Let V be the verifier in Theorem 7.1 when $k = \frac{3}{4\epsilon}$.

For any input x construct a label-cover instance as above using V. If $T = 2^{\log^{2k+2} n}$, the reduction runs in poly(T) time, and produces instances of size $O(T^2)$. Lemma 7.2 implies that if $x \in L$ there exists a labelling covering all the edges and otherwise no labelling covers more than $2^{-\log^k n}$ fraction of edges. Since $2^{\log^k n} = 2^{\log^{k/2k+4} T} \geq 2^{\log^{0.5-\epsilon} T}$, the gap is as claimed. $\Box$

7.1.1. Hardness of SV$_\infty$

Recall that the hardness result for SV$_\infty$ used Lemma 6.15. In this section we give some idea of how this result is proved; more details appear in [ABSS93].

The main idea is that Label Cover instances produced in the above reduction represent the following algebraic object. Let m be an integer and F a field. Let lines in F^m be defined as in Section 4.4.3.

In the bipartite graph (V_1, V_2, E) constructed by the reduction, V_1 is the set of all lines passing through points of a fixed set $S \subseteq \mathrm{F}^m$, where S contains an affine basis for F^m. The other side V_2 corresponds to points in F^m. An edge (v_1, v_2) is in E if the point of F^m represented by v_2 lies on the line represented by v_1. The set of labels and the edge relations Π_e also has a related algebraic description.

The result is proved using an *expansion*-like property of the graph (V_1, V_2, E), which we don't state here.

7.2. Unifying Label-Cover and MAX-3SAT(13)

In this section we state a recent result of Ran Raz from [Raz94]. An immediate consequence is that the inapproximability result for Label Cover can be derived from the inapproximability result for MAX-3SAT(13). Before this development we knew how to prove the hardness of Label Cover only by using the result in [FL92] – a result that seemed independent of the PCP Theorem.

Raz proves the so-called *parallel repetition conjecture*, a longstanding conjecture from the theory of interactive proofs. We describe only the consequence for restricted PCP verifiers.

Let V be any restricted verifier using alphabet Σ. Let us define $V \uparrow k$, the *kth product of* V, as follows. Verifier $V \uparrow k$ expects tables T_1', T_2' of size $|T_1|^k$ and $|T_2|^k$ respectively, where the set of locations in T_1' is in 1-to-1 correspondence with the set of all possible k-tuples of locations in T_1, and table T_2' bears a similar relation to T_2. The alphabet of $V \uparrow k$ is Σ^k. Verifier $V \uparrow k$ performs k independent runs of V, except it bunches up the sequence of k queries to T_1 into a single query to T_1', and the sequence of queries to T_2 into a single query to T_2. It reads the k-tuples of symbols from these locations in T_1', T_2', and accepts iff all k runs of V would have accepted.

For any fixed input, if there exist tables (T_1, T_2) which V accepts with probability 1, then there clearly exist tables (T_1', T_2') which $V \uparrow k$ accepts with probability 1.

Claim: (Raz) For a given input, suppose verifier V accepts every pair of tables T_1, T_2 with probability less than p where $p < 1$. Then $V \uparrow k$ will accept every pair of tables T_1', T_2' with probability less than $p^{\frac{ck}{\log N}}$, where c is fixed positive constant depending only upon the verifier (and not the input size) and N is the number of strings that V could give as answers. □

The proof of this claim is very complicated.

As an immediate consequence we can improve Theorem 7.1.

Theorem 7.3: *For all positive increasing functions k of the input size,*

$$NP \subseteq RPCP(k(n)\log n,\ k(n),\ k(n)).$$

Proof: Example 7.1 implies that NP $\subseteq$ RPCP$(\log n, 1, 1)$. Take the $O(k(n))$-th product of that verifier. Since the number of possible answers for the original verifier is $O(1)$ (actually 8), probability of incorrect acceptance becomes $2^{-ck(n)}$. Hence NP $\subseteq$ RPCP$(k(n)\log n,\ k(n),\ k(n))$. □

Implications for Label Cover. We saw in Lemma 7.2 that Label Cover captures exactly the combinatorial structure of RPCP verifiers. Raz's result implies that Label-Cover (max.version) is self-improvable under the following operation.

Given an instance (V_1, V_2, E, N, Π) of Label Cover (max. version), define its kth product as follows. The underlying graph is (V_1^k, V_2^k, E^k) (where V_1^k denotes cartesian product of the set V_1 with itself k times), and the set of labels is $[1, N^k]$. The set of labels is viewed actually as $[1, N]^k$, the set of k-tuples of labels. Thus a labelling must now assign a k-tuple of labels from $[1, N]$ to vertices in $V_1^k \cup V_2^k$. The new set of edge-functions, denoted Π^k, are defined as follows. Let e be a k-tuple of edges $(e_1, \ldots, e_k)$ in the original graph. Then define

$$\Pi_e^k(a_1^1, a_1^2, \ldots, a_1^k) = (a_2^1, a_2^2, \ldots, a_2^k)$$

iff the labels satisfy

$$\forall j \in 1, \ldots, k, \quad \Pi_{e_j}(a_1^j) = a_2^j.$$

Raz's result implies that if the value of the optimum in original instance is at most p, then the value of the optimum in the kth product of the instance is p^{ck}. In other words, so long as the original optimum is some constant less than 1, the new optimum decreases exponentially as we increase k.

We already gave a reduction from MAX-3SAT(13) to Label Cover (max. version) that shows the hardness of $(1 + \epsilon)$-approximation. As a consequence of the above result about self-improvement, this reduction can be modified (using the above notion of kth product) to give a hardness result for approximation within large factors (namely, the hardness result in Theorem 6.5) as well.

This leaves a tantalizing open problem: Is there a booster-like construction (in the sense of Section 6.5) for Label Cover, which can prove that n^δ-approximation is NP-hard, for some fixed $\delta > 0$? A recent result by Feige and Killian suggests that such boosters do not exist.

Chapter 8

Applications of PCP Techniques

The techniques used to prove the PCP Theorem have found other applications besides the hardness of approximations. This chapter surveys some such applications.

Many applications rely upon stronger forms of the PCP Theorem. We recall the setting of the PCP theorem, and then describe alternate settings that occur in its stronger forms.

The Basic Situation. A probabilistic polynomial time verifier needs to decide whether its input x satisfies a predicate Ψ, where Ψ is computable in nondeterministic polynomial time. The verifier is given random access to a remote database that purports to show that $\Psi(x) = 1$. In verifying the database, the verifier has to minimize the following two resources: the number of bits it examines in the data-base, and the amount of randomness it uses. What minimum amounts (as a function of input size n) of the two resources does it need, in order that the database have a reasonable chance of convincing it if $\Psi(x) = 1$, and a negligible chance otherwise?

According to the PCP theorem, upperbounds on the resources are: $O(\log n)$ random bits, and $O(1)$ query bits. (Further, if these amounts could be lowered any further, then P = NP ([AS92]).) Now we consider modifications of the Basic Situation. The first modification considers the complexity of constructing the database.

Situation 2. In the Basic Situation, suppose the input x is a 3CNF formula, and the predicate ψ is defined to be 1 iff x is satisfiable. Minimize the time required to construct the database, assuming the database constructor is already provided with a satisfying assignment. Also, give a way to construct the database such that the verifier can recover any desired bit of the satisfying assignment as efficiently as possible.

Situation 3. Similar to the Basic Situation, except that the predicate ψ is computable in nondeterministic time $t(n)$, where $t(n) = \Omega(\text{poly}(n))$. Make the verifier as efficient as possible.

8.1. Strong Forms of the PCP Theorem

We state stronger forms of the PCP Theorem that deal with the above two situations.

Theorem 8.1 (Strong Form 1): *In Situation 2, the database constructor runs in time poly(n). Further, it can construct the database in a way such that it contains an encoding of the assignment, which the verifier can decode bit-by-bit. Decoding a bit of the assignment requires examining $O(1)$ bits in the database. (The decoding algorithm is probabilistic.)*

Proof: (*Sketch*) Recall the constructive nature of the proof of the PCP Theorem. To obtain a database from a satisfying assignment one needs to construct the polynomial extension of the assignment, and the tables required by the various component procedures: Low-degree Test, the sum-check, the procedure that aggregates queries, and so on. A quick look at the descriptions of all these tables shows that they can be constructed in time poly(n). (Also, the Composition step of Chapter 3 is also quite constructive in nature.) Hence the first half of the Theorem is proved.

We give an indirect proof of the second half, specifically, the fact that the verifier can recover bits of the assignment from such a database. (A direct proof, using properties of the polynomial extension, is also possible. We do not give it here.)

Recall that in our construction the database has two parts. The first contains an encoding of the assignment. (The encoding uses polynomial extensions of bit-strings.) The second contains information showing that the assignment satisfies the formula. Now suppose the verifier wants to be convinced that the assignment satisfies a second NP-predicate ψ'. Then only the second part needs to be changed: the database constructor just adds information showing that the string encoded by the first part also satisfies ψ'.

In particular, the following predicate is computable in polynomial time (and therefore is an NP-predicate): YES on a string iff the ith bit of the string is 1. Denote this predicate by ψ_i.

The verifier stipulates that the second part of the database contain the following additional information: for each bit-position i in the assignment, if the ith bit is 1, a proof that the assignment satisfies ψ_i and otherwise a proof that the assignment satisfies $\overline{\psi_i}$, the negation of ψ_i.

If the verifier feels the need to decode the ith bit of the assignment, it can check (using $O(1)$ queries) a proof for ψ_i, or $\overline{\psi_i}$, as the case may be. If the check succeeds, then the ith bit has effectively been recovered. $\Box$

Polishchuk and Spielman ([PS94]) have further strengthened Strong Form 1. They show that the size of the database is just $O(n^{1+\epsilon})$, where n is the running time of the nondeterministic computation that computes Ψ, and ϵ is any positive constant.

Next, we state the result about Situation 3.

Theorem 8.2 (Strong Form 2): *([BFLS91, ALM+92]) In Situation 3, the verifier needs $O(\log t(n))$ random bits and $O(1)$ query bits. Its running time is poly$(\log t(n) + n)$, where n is the size of the input x. Further, the database continues to have the decoding property mentioned in Strong Form 1.*

Proof: (*Sketch*) First we describe the (trivial) modification to the proof of the PCP Theorem that has all the right properties, except the verifier runs in time poly$(t(n))$ instead of poly$(\log t(n) + n)$.

Using the obvious extension of the Cook-Levin theorem, do a reduction from predicate ψ (which is computable in nondeterministic time $t(n)$) to instances of 3SAT of size poly$(t(n))$. Then use the verifier of the PCP Theorem on such instances of 3SAT. It uses $O(\log t(n))$ random bits, queries $O(1)$ bits and runs in time poly$(t(n))$.

Now we describe how to make the verifier run in time poly$(\log t(n) + n)$. First, notice that the verifier of the PCP Theorem uses poly$(t(n))$ time solely because of Lemma 4.6 (the algebraic view of 3SAT). All other sub-procedures used to define the verifier contribute only time poly$(\log t(n))$ to the running time. (For instance, the technique of aggregating queries (Lemma 4.4) involves two simple algebraic procedures that run in time poly$(d + m)$, where m, d are respectively the degree of and the number of variables in the polynomial extensions being used. A similar statement holds for the sum-check. Recall that in all those cases, m and d are poly$(\log t(n))$.)

To improve Lemma 4.6 we need an idea from [BFLS91]. Use Levin's Theorem ([Lev73]) to reduce the decision problem on input x to an instance of *Tiling*. The *Tiling* problem asks for square unit-sized tile to be put on each vertex of a $K \times K$ grid, such that each tile is one of a set of possible types, and the set of tiles around each grid-point looks *valid*. (The first line of the grid is already tiled; the final tiling has to extend this.) The size K of the desired tiling is poly$(t(n))$, the number of allowable tile-types is c, and the number of valid neighborhoods allowed in the tiling is d, where c and d are some constants (independent of n). Levin's reduction runs in time poly$(n + \log t(n))$. (*An aside*: Levin's Theorem follows easily from the tableau viewpoint described earlier in Chapter 2.)

Modify the verifier of Lemma 4.5 to work with the Tiling problem instead of 3SAT. It now expects the database to contain a polynomial extension of a valid tiling. Modify the ideas of Lemma 4.6 to produce an algebraic view of the tiling problem. (There is no need to write the functions χ_j, s_j etc. of that Lemma now. Instead, there is a more direct way to write an algebraic formula that represents the set of valid neighborhoods. We don't give further details here.)

This allows the verifier to run in time poly$(t + \log n)$. $\Box$

Note: This theorem was proved in [BFLS91] in a somewhat weaker form; their verifier required $\text{poly}(\log t(n))$ random bits and as many query bits.

8.2. The Applications

This section describes various applications of the above Strong Forms.

8.2.1. Exact Characterization of Nondeterministic Time Classes

This application uses Strong Form 2.

Let $\text{Ntime}(t(n))$ for $t(n) \geq \text{poly}(n)$ denote the set of languages computable in nondeterministic time $t(n)$. The following is a restatement of Strong Form 2.

Theorem 8.3: *Ntime*$(t(n))$ = *PCP*$(\log t(n), 1)$ $\forall\, t(n) \geq$ *poly*(n).$\Box$

This characterization generalizes the result MIP = NEXPTIME in [BFL91], which can be equivalently stated as $\text{Ntime}(2^{\text{poly}(n)}) = \text{PCP}(\text{poly}(n), \text{poly}(n))$. It also generalizes the work of [BFLS91, FGL+91] whose result could be interpreted as saying that $\text{Ntime}(t(n)) \subseteq \text{PCP}(\text{poly}(\log t(n)), \text{poly}(\log t(n)))$.

8.2.2. Transparent Math Proofs

In this section we show that formal proofs in first order logic can be checked very efficiently by a probabilistic algorithm. The algorithm needs to examine only a constant number of bits in the proof. This application of Strong Forms 1 and 2 was suggested by Babai et al. ([BFLS91]).

We have to restrict ourselves to *reasonable* axiomatic systems over first order logic. These are axiomatic systems for which a Turing Machine can check proofs in polynomial time. More specifically, given any alleged theorem T and a claimed proof Π for it, the Turing Machine can determine in time $\text{poly}(|T| + |\Pi|)$ whether or not the proof is correct in the system. Most popular axiomatic systems (for instance, the Zermelo-Fraenkel axioms) are reasonable. They involve a constructible set of axioms and induction rules, and checking whether each step of a given proof is deduced correctly from preceding steps involves a simple syntactic check.

For a reasonable axiomatic system, let the term *proof-checker* refer to any Turing Machine (probabilistic or deterministic) that can check proofs in that system. The following theorem shows that the proof-checker can be made quite efficient.

Theorem 8.4: *For any reasonable axiomatic system, there is a probabilistic proof-checker Turing Machine that, given a theorem-candidate T and a proof-candidate Π, runs in time poly*$(|T| + \log |\Pi|)$, *examines only $O(1)$ bits in the proof, and satisfies the following.*

1. *If theorem-candidate T has a proof in the system of size N, then there exists a string Π of size poly(N) such that the checker accepts (T, Π) with probability* 1.

2. *If T has no proofs, the checker rejects (T, Π) with probability at least $3/4$, where Π is any string whatsoever.*

3. *The following two transformations can be done in polynomial time. (a) Transforming any string Π that is accepted by the verifier with probability at least $> 1/4$ (in particular, (2) implies that T in this case must be a theorem) to a proof in the axiomatic system. (b) Transforming any valid axiomatic proof to a proof that is accepted by the checker with probability* 1.

Notes: Condition (3) shows a polynomial-time equivalence between provability in the classical sense in the axiomatic system, and provability for our verifier. The advantage of our system is that the running time of the verifier grows as a polynomial in the logarithm of the proof-size. Also, only a constant number of bits are examined in the proof-string.

Proof: Use the well-known connection between mathematical proofs and nondeterministic Turing Machines. For every reasonable system, by definition there is a nondeterministic Turing Machine that accepts the language $\{(T, n) : T \text{ is a theorem that has a proof of size } n\}$. (Note: the machine guesses a proof of length n, and then checks in poly(n) time that it is correct.) Using Theorem 8.3, parts 1 and 2 of the theorem statement follow.

Further, part 3 (a) follows from the fact that the database is efficiently constructible. Part 3 (b) uses the "decoding property" mentioned in Strong Forms 1 and 2: Given a database that is accepted with high probability, the verifier can decode the original nondeterministic guess (in this case, a proof of theorem T) bit by bit.

$\square$

A philosophical problem needs to be pointed out here. It may appear that our construction has simplified the checking of math proofs, since our checker needs to examine only $O(1)$ bits in the proof. However, in another sense, the new checker's program is quite complex. At least the way we proved the above theorem, the checker must write down a 3SAT formula (or set of acceptable tile types) that expresses an axiomatic system. This is not an easy task, and certainly not so for humans.

8.2.3. Checking Computations

Strong Form 3 also enables constructions of *certificates* for all nondeterministic computations. The certificate's length is polynomial in the running time of the computation, and checking it requires examining only $O(1)$ bits in the certificate. (This application of PCP-type results was also suggested in [BFLS91].)

The situation we have in mind is a restatement of Situation 3. Suppose a user has a nondeterministic program P and an input x. The user would like to know if the nondeterministic program has an accepting computation on the input.

A database constructor with unlimited computational power (say, a supercomputer) can go through all possible branches of the nondeterministic program P, find an accepting branch (if one exists), and change it into a database that the verifier can check by examining only $O(1)$ bits in it. This database can therefore be viewed as an (easily checkable) certificate that P accepts x.

Of course, as a special sub-case, the above construction also applies to deterministic programs. More importantly, in many cases, a complicated deterministic program can be replaced by a trivial (but equivalent) nondeterministic program. Hence a certificate that the nondeterministic version accepts is also a certificate for the more complicated deterministic version. We illustrate this point with an example.

Example 8.1: Suppose we are given a number N. There is a well-known deterministic algorithm that checks in time $n^{O(\log \log n)}$ whether or not N is composite, where $n = \log N$. But this algorithm is complicated, and therefore we might have reasons to mistrust any software that claims to implement it.

Now consider the following nondeterministic program: Guess a number between 1 and N, and accept iff it is a nontrivial divisor of N. This program runs in time $O(n^2)$ or so and has an accepting branch iff N is composite.

Hence certificates for the nondeterministic program are much shorter – and simpler – than those for the deterministic program.

(Note, however, that finding a certificate for the nondeterministic computation is equivalent to finding a divisor of the number N, which is the celebrated *factoring* problem. The best known algorithms for it run in time $2^{\sqrt[3]{n}}$. Hence by insisting that it wants to see only certificates for the nondeterministic program, the verifier has made the task of the certificate constructor much more difficult.)

A caveat is in order. Our technique does not represent a way to check software. We assumed throughout that software for the nondeterministic program P is reliable.

8.2.4. Micali's Certificates for VLSI Chips

Micali [Mic94] notes that the above idea of checking computations, since it assumes that software is reliable, makes more sense in the context of checking VLSI chips. Chips are designed carefully (in other words, they are programmed with reliable software), but the fabrication process might introduce bugs in the hardware. Instead of testing the chip exhaustively, we could require (by adding the necessary hardware to the chip) that it always give us a certificate that its computation was done according to specifications. However, input/output from chips is slow, so efficiency would be terrible if we use the large certificates described above (since their size polynomial in the running time of the computation). Micali shows how to hash down the certificate to polylogarithmic size in a cryptographic fashion. By "cryptographic hashing" we mean that although it is possible to produce a fraudulent certificate, doing so takes a lot of time (which the chip does not have).

Micali's idea works also with the weaker result about checking nondeterministic computations that appears in [BFLS91], but efficiency is better using the PCP theorem. Also, he points out an interesting extension of the class IP. (Section 2.5).

8.2.5. Characterization of PSPACE (Condon et al.)

Condon, Feigenbaum, Lund and Shor ([CFLS93]) give a new characterization of PSPACE, the set of languages accepted by machines that run in polynomial space. Their result PSPACE = $RPCDS(\log n, 1)$ uses the Strong Form 1 of the PCP Theorem.

$RPCDS(r(n), 1)$ is a class of languages defined using a *debate* between two players, $\exists$ and $\forall^+$, where $\forall^+$ is just a source of independent random bits. The players alternate in setting down strings of bits on a debate-tape, which is checked at the end by a polynomial time verifier, who accepts or rejects. (The verifier has random access to the debate tape.) Language L is in $RPCDS(r(n), 1)$ if the verifier on inputs of size n uses $O(r(n))$ random bits, examines $O(1)$ bits in the tape, and satisfies: For all inputs in L, there is an $\exists$ player such that the debate is accepted with probability 1, and for all inputs not in L the debate is rejected with probability at least 1/2 (irrespective of what the $\exists$ player does).

Shamir's result ([Sha92]) implies that PSPACE = $RPCDS(0, \text{poly}(n))$. Condon et al. obtain their improvement by stipulating that the player $\exists$ write down at the end of the debate a "certificate" (that is, the database referred to in Strong Form 1) showing that the verifier would have accepted the debate. Recall that this means that the "certificate" contains an encoding (using polynomial extensions) of the debate, lots of other tables and so on. Strong Form 1 implies that the verifier can check this certificate by examining only $O(1)$ bits in it. Only one task remains: how to verify that the debate encoded in such a certificate is the actual debate that took place. In the [CFLS93] paper it is shown (using Shamir's result) that the verifier only needs to decode $O(1)$ bits from the encoded debate (the decoding requires reading only $O(1)$ bits, according to Strong Form 1), and check them off against the corresponding bits in the actual debate.

Condon et al. use their characterization of PSPACE to show the hardness of approximating some PSPACE-hard problems.

8.2.6. Probabilistically Checkable Codes

Recall the definition of a *code* from Section 3.1. For now we restrict attention to codes on the alphabet $\{0,1\}$.

Definition 8.1: A family of *Probabilistically Checkable Codes* is a family of codes, (i.e., with one code for each codeword size), whose minimum distance $\delta_{\min}$ is some constant (like 0.1) indenpendent of the codeword size. Further, the family has an associated polynomial time randomized checker. Given a word of size n, the checker uses $O(\log n)$ random bits, examines $O(1)$ bits in the word, and has the following properties.

1. If the word is a codeword then the checker accepts with probability 1.
2. If the word is not $\delta_{\min}/3$-close then the checker rejects with probability 1/2.

Proposition 8.5 (ALMSS): *For some $c < 1$ there exist probabilistically checkable codes in $\{0,1\}^n$ that contain 2^{n^c} strings.*

We can use stronger versions of Definition 8.1, none of which affect the validity of the previous proposition: the codes have associated coding/decoding algorithms that run in polynomial time; the verifier runs in time poly$(\log n)$ instead of poly(n); a probabilistic decoding of any bit in a $\delta_{\min}/3$-codeword requires examining only $O(1)$ bits in the word, and so on.

We will not prove Proposition 8.1 here. The construction uses techniques from the proof of the PCP Theorem. Recall that the proof of the PCP theorem consisted of a sequence of encoding schemes for assignments (see Figure 4.3 for a bird's-eye view). The same sequence of schemes works also for encoding bit-strings.

8.2.7. Kilian's Communication-efficient Zero-Knowledge Arguments

We will not attempt to give an exact definition of zero-knowledge arguments here. Roughly speaking, the situation involves two parties – both of whom run in probabilistic polynomial time – and a 3SAT formula. One party has a satifying assignment to the formula, and wants to convince the other of this fact in such a way that the other party does not learn even a single bit in the satisfying assignment.

Protocols for doing this are known, but they require too much communication between the parties. Kilian ([Kil92]) shows how to reduce the communication requirement. His idea is to hash down, in a cryptographic fashion, the "probabilistically checkable" database of Strong Form 1. (Recall that Micali's idea is similar.)

He needs something more than the Strong Form 1, specifically, the fact (proved in [BFLS91, PS94]) that the database given by Strong Form 1 has size $n^{1+\epsilon}$, where n is the size of the 3SAT formula.

8.2.8. Khanna et al.'s Structure Theorem for MAX-SNP

The class APX contains optimization problems which can be approximated within some constant factor in polynomial time. An open question posed in [PY91] was: is APX contained in MAX-SNP? The answer turns out to depend upon how MAX-SNP is defined. It appears that the definition intended in [PY91] (although never stated explicitly thus) was that every problem that has an approximation-preserving reduction to MAX-3SAT should be considered to be in MAX-SNP. With this definition, MAX-SNP equals APX ([KMSV94]). The proof is not too difficult: the PCP Theorem can be used to give a reduction from any APX problem to MAX-3SAT.

8.2.9. The Hardness of finding Small Cliques

Given a graph of size n, how hard is it to determine whether or not it has a clique of size $\lceil \log n \rceil$? The trivial algorithm based upon exhaustive search takes $O(n^{\log n})$ time. Does a polynomial time algorithm exist?

This question was raised in [PY93a], and is also related to the study of fixed parameter intractability ([DF95]).

Recently, Feige and Killian ([FK94a]) related this question to another question about traditional complexity classes. They show that if a polynomial time algorithm exists, then $\mathrm{Ntime}(t(n)) \subseteq \mathrm{Dtime}(2^{t(n)^{1-\epsilon}})$ for some small positive constant ϵ.

They use the version of Strong Form 1 due to [PS94] (see the note following Strong Form 1.) Their idea is to do a reduction to clique using this strong form, and then apply a booster-like construction as in Section 6.5.

Note: Noam Nisan has since shown the same result without using the PCP Theorem.

Chapter 9

Open Problems

We list two types of open problems. The first type, contained in Section 9.1, concern the hardness of approximation. The second type, in Section 9.2, concern "PCP techniques." The latter term is an umbrella term for ideas like program-checking, self-testing/correcting, random self-reducibility, and algebraic properties of polynomials – in other words, the ingredients of the new results in complexity theory.

Finally, in Section 9.3 we discuss the following open problem: Does the PCP Theorem have a simpler proof?

9.1. Hardness of Approximations

There are two major open areas here: First, to show the inapproximability of problems for which this is not known. Second, to improve existing inapproximability results.

9.1.1. Proving hardness where no results exist

Despite great progress on proving hardness results for problems like clique, chromatic number, set cover, etc., similar results elude us for the following problems.

Shortest Lattice Vector: Given an integer lattice $\left\{\sum_i \alpha_i \hat{b}_i : \alpha_i \in \mathbf{Z}\right\}$, the problem is to find a non-zero lattice vector whose ℓ_2 norm is the smallest (see Section 6.4). It is open even whether exact optimization is NP-complete. The best factor of approximation currently achievable in polynomial time is $2^{O(n/\log n)}$ ([Sch85]). The best inapproximability result says that the version of the problem using the ℓ_∞ norm is almost-NP-hard upto a factor $2^{\log^{0.5-\epsilon} n}$ ([ABSS93], also Section 6.4). Improving the ℓ_∞ result to a factor of $\sqrt{n}$ will prove hardness of the ℓ_2 version as well, since the optima in the two norms are related by a factor of $\sqrt{n}$.

The rigid structure of the lattice makes it difficult to come up with reductions. The geometric arguments used in the ℓ_∞ result may provide a hint on how to proceed.

Euclidean TSP. This is the version of the Travelling Salesman Problem where points lie in a Euclidean space. Exact optimization is NP-hard even when all points lie in a plane ([GGJ76, Pap77]). The best factor of approximation currently achievable in polynomial time is 3/2 ([Chr76]). No inapproximability results exist. It is known however that Metric TSP, the version in which the underlying space is a (possibly non-Euclidean) metric space, is MAX-SNP-hard ([PY93b]), so as a consequence of Theorem 6.9, finding $(1+\epsilon)$-approximations is NP-hard.

The NP-hardness of exact optimization in the Euclidean case is proved using the NP-completeness of planar-SAT. But MAX-SAT restricted to planar instances has a polynomial time approximation scheme ([AKM94]), which rules out the use of planar-SAT in proving hardness of approximation.

Instances of SAT produced by the current proof of the PCP theorem represent high-dimensional objects (namely, a geometry involving the points and lines of a $\log n$-dimensional space; see chapter 5, and also Lemma 4.4). It seems difficult to do reductions from these to planar TSP, but perhaps a reduction is possible to higher-dimensional TSP.

Edge-deletion type problems. This group of problems, proposed by Yannakakis ([Yan81]) consists of any problem stated in the following form, for some property T of graphs that is closed under edge-deletion (for example "disconnectedness"): Remove the minimum possible number of edges so that the remaining graph satisfies T.

The following are two well-known examples: Graph bisection (T = there is a set of connected components which together include exactly $n/2$ vertices), and Minimum-feedback Arc Set (T = acyclicity).

A series of papers starting with ([LR88]) use flow techniques to approximate many edge-deletion problems within factors like $\log n$, or $\text{poly}(\log n)$. (For graph bisection, the approximate solution produces not an exact bisection, but a $1/3 : 2/3$ split of the vertex set.)

No good hardness results are known; in many cases (like graph bisection) not even MAX-SNP-hardness is known. The following clean problem seems to be a good candidate to prove hard to approximate: Given an instance of MAX-2SAT, delete the smallest number of clauses so as to make it satisfiable. (To see why this fits the edge-deletion framework, and also an approximation algorithm for a related problem, refer to [KARR90, GVY93].)

9.1.2. Improving existing hardness results

There are two ways to improve existing hardness results. First, to base the result on the assumption $\text{P} \neq \text{NP}$ (many results are currently based upon stronger assumptions).

Second, to show that approximation within larger factors is also hard. For most problems there is a large gap between those factors of approximation known to be NP-hard, and those achievable in polynomial time.

Basing results on P $\neq$ NP.

One shortcoming of many existing hardness results is that they are based upon complexity assumptions stronger than "P $\neq$ NP." This was also originally the case with the clique problem ([FGL$^+$91]), but as a result of ([AS92]) we are now able to base that hardnes result upon P $\neq$ NP. So there is hope that the same may be possible with other problems.

The reason for resorting to strong complexity assumptions is that many hardness results involve reductions from Label Cover (for a definition see Chapter 6). Approximating Label Cover upto a factor of $2^{\log^{0.5-\epsilon} n}$ is known to be only almost-NP-hard, instead of NP-hard. (Approximating within constant factors is NP-hard, however.) Thus reductions from Label Cover also prove almost-NP-hardness of large factor approximation. On the other hand, a proof that n^ϵ-approximation to Label Cover is NP-hard (for instance) would immediately make Label Cover much more useful as a canonical problem. One way to prove such a result is to prove the following conjecture (for a definition of RPCP see Definition 7.2).

Conjecture 9.1: *For all* $k \leq O(\log n)$

$$NP \subseteq RPCP(\log n, k, k).$$

□

The conjecture is true for the case $k = O(1)$ ([FK94b])[1]. In general, if conjecture 9.1 holds for any given k, then approximating Label Cover within a factor 2^k is NP-hard.

Approximation Problems for which Conjecture 9.1 implies NP-hardness.

1. Set-Cover upto a factor $O(\log n)$. For this result it suffices that the conjecture be true for $k = O(\log\log n)$([LY94]), or even that the conjecture be true with $O(1)$ tables instead of just 2 ([BGLR93]).

2. Lattice and other Problems. An entire group of problems (involving lattices, systems of linear equations and inequalities) from section 6.4, upto a factor of n^ϵ for some small ϵ.

3. Vertex Deletion problems. An entire family of problems, upto a factor of n^ϵ ([LY93]).

[1] A very recent result by Feige and Killian suggests that the conjecture is hard to prove.

In addition, the *longest path* problem is currently known to be NP-hard to approximate within any constant factor. Assuming SAT cannot be solved in subexponential time the factor can be pushed up to $2^{\log^{1-\epsilon} n}$ ([KMR93]). It is not clear how to base the second result on P $\neq$ NP, even if conjecture 9.1 is true, since the known reduction (from MAX-3SAT(4)) inherently blows up the size of the instance beyond any polynomial. (See also the note in Section 6.5.)

Improving the Factors

The other way to improve known hardness results would be to prove that approximation is hard even for larger factors. The following is a list of some of the significant problems.

Clique and Independent Set. Assuming NP problems cannot be solved probabilistically in subexponential time, $\sqrt[4]{n}$-approximation to Clique and Independent Set is impossible in polynomial time ([BS94]). The best polynomial-time algorithms achieve a factor $n/\text{poly}(\log n)$. Can we prove hardness for a factor $n^{1-\epsilon}$?

Chromatic Number. (i) The discussion from independent set applies to chromatic number too. (ii) Given a 3-colorable graph, what is the least number of colors with which we can color it in polynomial time? The best algorithms use $n^{0.25}$ colors ([KMS94]). The best hardness result says that at least 5 colors are needed if P $\neq$ NP ([KLS93]). Can "5" be improved to n^{ϵ}, for some small enough ϵ, as many believe? A result by A. Blum shows that if coloring 3-colorable graphs with even poly($\log n$) colors is hard, then approximating Clique within a factor $n^{1-\delta}$ is hard, where δ is an arbitrarily small positive constant. (A relevant fact here – which Blum also uses in his result – is that chromatic number is "self-improvable", as shown in [LV89].)

Classic MAX-SNP-hard problems. These include vertex cover, TSP with triangle inequality, MAX-SAT, etc. The best polynomial time algorithms achieve approximation factors of 2, 3/2 and $4/3-0.01$ respectively. We only know that $(1+\epsilon)$-approximations are hard for $\epsilon \approx 0.01$. Can the hardness result be improved? A surprising development in this area is the result of Goemans and Williamson [GW94], where it is shown that MAX-2SAT and MAX-CUT, two other MAX-SNP-hard problems with with classic 2-approximation and 4/3-approximation algorithms respectively, can be approximated within a factor better than 1.13.

9.1.3. Obtaining Logical Insight into Approximation Problems

In Section 6.6 we gave a survey of known inapproximability results. Problems seem to fall into four main classes, according to the factor of approximation which is provably hard to achieve in polynomial time.

Why do problems fall into these four classes? Is there a method (at least at an intuitive level) for recognizing, for a given problem, which of these classes it falls in?

No satisfactory answers to such questions are known any class except the first. (Recall that this class contains only MAX-SNP-hard problems.)

Further, the edge-deletion problems seem to form a class of their own. It would be nice to find a "complete" problems in this class, in other words, a problem whose inapproximability implies the inapproximability of the entire class.

9.2. Open Problems connected with PCP techniques

The class PCP($r(n), r(n)$) for $r = o(\log n)$. The class PCP($o(\log n), o(\log n)$) is contained in NP , but does not cotain any NP-complete problems if P $\neq$ NP ([AS92]). A result in [FGL$^+$91] shows how to reduce the question of membership in a language in PCP($r(n), r(n)$) to an instance of Clique of size $2^{O(r(n))}$. So the membership problem seems to involve limited nondeterminism [PY93a]) but it is open if there is an exact characterization lurking there.

Size of the proof. In the proof of the PCP theorem, what is the minimum size of the proof needed for 3SAT formulae of size n? In our proofs we were sloppy with the numbers, but the best size that achievable is $n^{2+\epsilon}$ ([Sud92]). A tighter construction ([PS94]) achieves size $n^{1+\epsilon}$. Can we achieve size $n\,\mathrm{poly}(\log n)$? The size is important for cryptographic applications [Kil92].

Size of probabilistically checkable codes. These codes were defined in Section 8.2.6. The best construction ([PS94]) achieves a constant minimum distance (say, $\delta_{\min} = 0.01$) and encodes n bits with $n^{1+\epsilon}$ bits. Can the size of the encoding be reduced to $n\,\mathrm{poly}(\log n)$, or better still, to $O(n)$? Shannon's theorem says the size would be $O(n)$ if we didn't impose the probabilistic checkability condition [MS77]).

Improving the low-degree test. Does the low-degree test work even for high error rates? In other words, is Theorem 5.1 in Chapter 5 true even when the success rate is less than 0.5 (say), and $|\mathrm{F}| = \mathrm{poly}(d)$?

Self-correction on polynomials. Can polynomials be self-corrected (for definition see [BLR90]) in the presence of high error-rates? A *self-corrector* is a probabilistic program that is given a rational number $p > 0$ and a function $f : \mathrm{F}^m \to \mathrm{F}$ that is $(1-p)$-close to $\mathrm{F}_d[x_1, \ldots, x_m]$. The self-corrector's task, given an arbitrary point b in F^m, is to produce $g(b)$ in time $\mathrm{poly}(d, 1/p, \log|\mathrm{F}|)$, where $g \in \mathrm{F}_d[x_1, \ldots, x_m]$ is any polynomial that agrees with f in a fraction p of the points. Self-correctors are known to exist for all p such that $p \geq 0.5 + \epsilon$ for some $\epsilon > 0$. The case $p \leq 0.5$ is open. In general we seem to be missing some crucial insight into $(1-p)$-close functions for $p \leq 0.5$, which is possibly why the previous problem is also open.

Applications to cryptography. Do the algebraic techniques of the PCP results have applications in cryptography and program checking? For instance, a key result in cryptography is the construction of a hard-core bit for pseudo-random generation

([GL89]). At the heart of this result is a simple self-corrector (in the sense of [BLR90]) for linear functions over GF(2) (these functions were also encountered in Section 4.3). Now we know of much stronger results about polynomials (e.g., those in Chapter 5). What are the applications (if any) for cryptography? One possible application could be pseudo-random generation in parallel, a longstanding open problem.

Implications for complexity classes. We now know of PCP-like characterizations not only for non-deterministic time classes (Section 8.2.1) but also for PSPACE and PH (see Section 8.2.5). What are the implications of these new characterizations, if any? Can new characterizations be given for any other complexity classes, say P or EXP-TIME?

9.3. Does the PCP theorem have a simpler proof?

We mentioned earlier, while describing the overall picture of the proof of the PCP theorem in Section 4.5, that central to our proof of the PCP Theorem is a new way to encode satisfying assignments. The encoding uses polynomials (represented *by value* instead of by coefficient). Specifically, it uses the fact that the set of low-degree polynomials, when represented by value, form a code of large minimum distance (see Section 4.1). The definition of the encoding is not simple; it involves many steps, where each step consists of defining a new verifier. The verifiers are composed at the end to give a final verifier (and a final encoding). Figure 4.3 gives a bird's-eye view of this process.

Must every proof of the PCP Theorem be this complicated? There is no easy answer. In fact it is not even clear why the proof needs to involve an encoding process. In Claim 2 given below, we try to give intuition why. Further, we try to explain at an intuitive level why the encoding must make use of error-correcting codes (as ours did). We actually argue something stronger: that the encoding must use an object very much like a probabilistically checkable code[2] (see Definition 8.1).

We state two claims, the first rigorous, and the second intuitive.

The first Claim makes the following intuition precise: If a PCP($\log n, 1$) verifier accepts one string but rejects the other, then the two strings must differ in many bits.

Claim 1. *Every PCP*($\log n, 1$) *verifier V, has an equivalent* uniform form, *in which it has the following property. There is a positive integer C such that for every two proof-strings π_1 and π_2 and any input x,*

$$|\Pr[V \textit{ accepts } \pi_1 \textit{ on input } x] - \Pr[V \textit{ accepts } \pi_2 \textit{ on input } x]| \le C \cdot \delta(\pi_1, \pi_2) \qquad (9.1)$$

where $\delta(\pi_1, \pi_2)$ is the Hamming distance between π_1 and π_2 (as defined in Section 3.1).

Proof (Sketch). Note that the verifier will not give different answers on two strings unless

[2]Our argument in this section is somewhat imprecise. It is made more precise in [Aro94].

if it queries a bit-position in which they differ. In other words, for any strings π_1 and π_2, an upper bound on the quantity

$$|\Pr[V \text{ accepts } \pi_1 \text{ on input } x] - \Pr[V \text{ accepts } \pi_2 \text{ on input } x]|$$

is given by $\Pr[V$ queries a bit-position in which π_1 and π_2 differ$]$. We show how to make the queries of the verifier uniformly distributed, so that the above upper bound can be given in terms of the distance between π_1 and π_2.

Let $K = \text{poly}(n)$ be the size of the proof-string, and $q = O(1)$ be the number of bits the verifier queries in a single run. Consider the following probability distribution.

$$p_i = \Pr[\ V \text{ queries bit-position } i \text{ in the provided proof in its first query}]. \tag{9.2}$$

We first modify the verifier so that this distribution becomes uniform, and furthermore, is identical for all q queries. (*Important: The queries could be correlated; just their distribution is uniform.*) Then the claim follows, since if

$$\Pr[\text{the first bit queried in } \pi_1 \text{ and } \pi_2 \text{ is different}] = p,$$

then

$$|\Pr[V \text{ accepts } \pi_1 \text{ on input } x] - \Pr[V \text{ accepts } \pi_2 \text{ on input } x]| \leq qp.$$

Finally, the uniformity of the above distribution implies $p = \delta(\pi_1, \pi_2)$. By substituting $C = q$ the claim is proved.

Now we explain the modification to V to achieve uniformity. First make the above distribution identical for all q queries by randomly scrambling the order in which V makes its q queries (recall: V queries the proof nonadaptively). To do this the verifier requires a random permutation of a set of C elements, in other words, $O(1)$ random bits. Now pick a sufficiently large integer $R = \text{poly}(n)$. Modify V so that it expects a proof of size RK, containing $\lfloor Rp_i \rfloor$ copies of the ith bit of the old proof.

Whenever V reads the ith bit in its proof, the modified verifier will choose two bits randomly from among all $\lfloor Rp_i \rfloor$ copies of this bit, check that they have the same value, and if so, use that common value.

The distribution induced by this query pattern is almost uniform when

$$R \geq \ \text{Number of choices for the original verifier's random string.}$$

□

Claim 2 (Intuitive): *Whenever we construct a* $PCP(\log n, 1)$ *verifier* V_1 *for 3SAT, we must implicitly define for every 3SAT instance* φ*, a one-to-many map* σ *from assignments (for* φ*) to sets of proof-strings such that*

- *If assignment A satisfies φ, there is a proof-string $\pi \in \sigma(A)$ such that*

$$\Pr[V_1 \text{ on input } \varphi \text{ accepts } \pi] = 1.$$

- *If any proof-string π satisfies*

$$\Pr[V_1 \text{ on input } \varphi \text{ accepts } \pi] \geq \frac{3}{4}$$

then π has a unique pre-image under σ. Further, the pre-image is a satisfying assignment.

Justification: The construction of such a V_1 is an implicit proof of NP-completeness of the following language.

$$L = \left\{\varphi : \varphi \text{ is a 3CNF formula and } \exists \pi \text{ s.t. } \Pr[V_1 \text{ accepts } \pi \text{ on input } \varphi] \geq \frac{3}{4}.\right\}$$

(To see that this problem is in NP, look at the proof of Corollary 2.3.)

All known NP-completeness results map witnesses to witnesses in a one-to-many fashion[3]. We quote this as intuitive justification for our claim. □

Suppose we believe in Claim 2. Then we show how to use any uniform form PCP($\log n, 1$) verifier for 3SAT to define a code (over the alphabet $\{0,1\}$) that is quite close to being probabilistically checkable. Fix the input φ. Define the code as

$$\{\pi : \Pr[\text{the verifier on input } \varphi \text{ accepts } \pi] = 1\}.$$

The checker for this code is the uniform form verifier. It accepts all codewords with probability 1. Conversely, if it rejects any word with probability more than $1/2$, then, since it examines only $O(1)$ bits in the word, Claim 1 implies that the word is not δ-close, for some small enough constant δ.

We cannot rigorously prove that the code has minimum distance c for some fixed $c > 0$ (independent of φ). However, we can prove it under the assumption that the proof-strings in the code are in one-to-one correspondence with satisfying assignments. For, let π_1, π_2 be two codewords (representing different satisfying assignments) whose mutual distance is less than c. Let π' be any word that that agrees with both of them in $1 - c$ fraction of points. Then the probability that the verifier accepts π' is (by Claim 1) "close" to 1. By Claim 2, π' must be decodable to a unique pre-image. This contradicts the assumption that π_1, π_2 represented distinct satisfying assignments.

[3]The well-known randomized reduction from NP to UNIQUE-SAT ([VV86]) does not map every witness to a witness. We do not consider this reduction a counterexample because it succeeds with probability less than $1/n$, which in the PCP context is negligible. A deterministic (or low-error) version of this reduction would be a valid counterexample, though.

Conclusion. We have tried to argue that constructing probabilistically checkable codes (PCC's) is a pre-condition to proving the PCP Theorem. Currently the only way to construct such codes involves a small modification of the proof of the PCP Theorem: just take out the sum-check part from the first step of Figure 4.3. We feel that a simpler construction of PCC's will very likely yield a simpler proof of the PCP Theorem.

Finally, note the following "machineless" analogue of a PCC (in the spirit of Lemma 2.2, which replaced a $(\log n, 1)$-restricted verifier with a 3CNF formula). Any PCC yields a 3CNF formula in n variables such that for some constants $c, d > 0$: (i) the set of satisfying assignments form a code with minimum distance c (ii) the set of words that are not $\frac{c}{3}$-close satisfy fewer than $(1 - d)$ fraction of clauses. (Ideally, we want the formula to also satisfy the condition that the number of satisfying assignments is at least 2^{n^ϵ} for some $\epsilon > 0$.)

Currently, we do not know how to prove the existence of such 3CNF formulae, except as a by-product of the PCP Theorem. Hence an alternative proof of existence (say, a non-constructive proof) would also yield fresh insight into PCC's.

Bibliography

[ABSS93] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes and linear equations. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 724–733, 1993.

[ADP77] G. Ausiello, A. D'Atri, and M. Protasi. On the structure of combinatorial problems and structure preserving reductions. In *Proc. 4th Intl. Coll. on Automata, Languages and Programming*, 1977.

[ADP80] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21:136–153, 1980.

[AFK89] M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. *Journal of Computer and System Sciences*, 39, 1989.

[AFWZ93] N. Alon, U. Feige, A. Wigderson, and D. Zuckerman. Derandomized graph products. Manuscript, 1993.

[AK93] E. Amaldi and V. Kanna. The complexity and approximability of finding maximum feasible subsystems of linear relations. Technical report, TR # ORWP-11-93, Dept. of Mathematics, Swiss Federal Institute of Technology, Lausanne, 1993.

[AKM94] S. Arora, S. Khanna, and R. Motwani. A PTAS for planar MAX-SAT. Manuscript, 1994.

[AL95] S. Arora and C. Lund. Hardness of approximations. Survey chapter to appear in a book on Approximation Algorithms, D. Hochbaum, ed.. Available from the authors., 1995.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 13–22, 1992.

[AMS+92] S. Arora, R. Motwani, M. Safra, M. Sudan, and M. Szegedy. PCP and approximation problems. Manuscript, 1992.

[AMSP80] G. Ausiello, A. Marchetti-Spaccamela, and M. Protasi. Toward a unified approach for the classification of NP-complete optimization problems. *Theoretical Computer Science*, 12:83–96, 1980.

[Aro94] S. Arora. Reductions, codes, PCPs, and inapproximability. Unpublished manuscript, 1994.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 2–13, 1992.

[Bab85] L. Babai. Trading group theory for randomness. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 421–429, 1985.

[Bab86] L. Babai. On Lovász's lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–14, 1986.

[Bab94] L. Babai. Transparent proofs and limits to approximations. In *Proceedings of the First European Congress of Mathematicians*. Birkhauser, 1994.

[BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of the 7th Symp. on Theoretical Aspects of Computing*, pages 37–48. Lecture Notes in Computer Science, **415**, 1990.

[BF91] L. Babai and L. Fortnow. Arithmetization: a new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFLS91] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 21–31, 1991.

[BGKW88] M. Ben-or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi prover interactive proofs: How to remove intractability assumptions. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 113–121, 1988.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient multi-prover interactive proofs with applications to approximation problems. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 113–131, 1993.

[BJL+91] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 328–336, 1991.

[BK89] M. Blum and S. Kannan. Designing programs that check their work. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 86–97, 1989.

[BLR90] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 73–83, 1990.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[BM88] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, pages 254–276, 1988.

[BN90] J. Bruck and M. Naor. The hardness of decoding linear codes with preprocessing. *IEEE Transactions on Inform. Theory*, pages 381–385, 1990.

[Bol86] B. Bollobás. *Combinatorics*. Cambridge University Press, 1986.

[BP89] M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176, 1989.

[BR93] M. Bellare and P. Rogaway. The complexity of approximating non-linear programs. In P.M. Pardalos, editor, *Complexity of Numerical Optimization*. World Scientific, 1993. Preliminary version: IBM Research Report RC 17831 (March 1992).

[BS92] P. Berman and G. Schnitger. On the complexity of approximating the independent set problem. *Information and Computation*, 96(1):77–94, 1992.

[BS94] M. Bellare and M. Sudan. Improved non-approximability results. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 184–193, 1994.

[BW] E. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent Number 4,633,470.

[CFLS93] A. Condon, J. Feigenbaum, C. Lund, and P. Shor. Random debaters and the hardness of approximating stochastic functions. In *Proc. of the 9th Structure in Complexity Theory Conference*, pages 280–293, 1993. Also available as DIMACS Techreport TR 93-79.

[Chr76] N. Christofides. Worst case analysis of a new heuristic for the travelling salesman problem. Technical report, Grad. School of Industrial Optimization, Carnegie-Mellon University, 1976.

[CK94] P. Crescenzi and V. Kann. A compendium of NP optimization problems. manuscript, 1994.

[CL89] A. Condon and R. Ladner. On the complexity of space bounded interactive proofs. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 462–467, 1989.

[Con93] A. Condon. The complexity of the max-word problem and the power of one-way interactive proof systems. *Computational Complexity*, 3:292–305, 1993.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[DF95] R. Downey and M. Fellows. Fixed-parameter tractability and completeness ii: Completeness for W[1]. *Theoretical Computer Science*, 1995. to appear.

[DJP+92] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiway cuts. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 241–451, 1992.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In Richard Karp, editor, *Complexity of Computer Computations*, pages 43–73. AMS, 1974.

[Fei93] J. Feigenbaum. Locally random reductions in interactive complexity theory. In *Advances in Computational Complexity*, pages 73–98. American Mathematical Society, Providence, 1993. DIMACS Series on Disc. Maths. and Theoretical CS, volume 13.

[FF93] J. Feigenbaum and L. Fortnow. On the random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22:994–1005, 1993. Prelim. version in Proc. of the 6th Annual Structure in Complexity Theory Conference, 1991.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 2–12, 1991.

[FK94a] U. Feige and J. Kilian. Towards subexponential algorithms for NP. Manuscript, October, 1994.

[FK94b] U. Feige and J. Kilian. Two prover protocols–low error at affordable rates. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 172–183, 1994.

[FKN90] J. Feigenbaum, S. Kannan, and N. Nisan. Lower bounds on random-self-reducibility. In *Proceedings of the 5th Structure in Complexity Theory*, pages 100–109, 1990.

[FL92] U. Feige and L. Lovász. Two-prover one-round proof systems: Their power and their problems. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 733–741, 1992.

[Fre79] R. Freivalds. Fast probabilistic algorithms. In *LNCS 74*, pages 57–69. Springer Verlag, 1979.

[FRS88] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. In *Proceedings of the 3rd Conference on Structure in Complexity Theory*, pages 156–161, 1988.

[FSH94] K. Friedl, A. Shen, and Z. Hatsagi. The low-degree test. In *Proc. 5th SIAM Symposium on Discrete Algorithms*, 1994.

[FT85] A. Frank and É. Tardos. An application of simultaneous approximation in combinatorial optimization. In *Proc. 26th IEEE Symp. on Foundations of Computer Science*, pages 459–463, 1985.

[Für94] M. Fürer. Improved hardness results for approximating the chromatic number. Manuscript, 1994.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–425, 1981.

[GGJ76] M.R. Garey, R.L. Graham, and D.S. Johnson. Some NP-complete geometric problems. In *Proc. 8th ACM Symp. on Theory of Computing*, pages 10–22, 1976.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[GL89] O. Goldreich and L.A. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 25–32, 1989.

[GLR+91] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 32–42, 1991.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proofs. *SIAM J. of Computation*, 18:186–208, 1989.

[GMS87] O. Goldreich, Y. Mansour, and M. Sipser. Interactive proof systems: Provers that never fail and random selection. In *Proc. 28th IEEE Symp. on Foundations of Computer Science*, pages 449–461, 1987.

[Gol94] O. Goldreich. Probabilistic proof systems. Technical Report RS-94-28, Basic Research in Computer Science, Center of the Danish National Research Foundation, September 1994. (*To appear in the Proceedings of the International Congress of Mathematicians, 1994. Birkhauser Verlag.*).

[GS86] S. Goldwasser and M. Sipser. Private versus public coins in interactive proof systems. In *Proc. 18th ACM Symp. on Theory of Computing*, pages 59–68, 1986.

[GS92] P. Gemmell and M. Sudan. Highly resilient correctors for polynomials. *Information and Computation*, 43:169–174, 1992.

[GVY93] N. Garg, V.V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)-cut theorems and their applications. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 698 –707, 1993.

[GW94] M. Goemans and D. Williamson. A 0.878 approximation algorithm for MAX-2SAT and MAX-CUT. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 422–431, 1994.

[HS92] K-U. Hoeffgen and H-U. Simon. Robust trainability of single neurons. In *Proceedings of the Conference of Learning Theory*, pages 428–439, 1992.

[Joh74] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[Joh92] D. S. Johnson. The NP-completeness column: an ongoing guide. *Journal of Algorithms*, 13:502–524, 1992.

[JP78] D. S. Johnson and F. P. Preparata. The densest hemisphere problem. *Theoretical Computer Science*, 6:93–107, 1978.

[Kan87] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3), 1987.

[Kan92] V. Kann. *On the approximability of NP-complete optimization problems.* PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 1992.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In Miller and Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KARR90] P. Klein, A. Agarwal, R. Ravi, and S. Rao. Approximation through multicommodity flow. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pages 726 –727, 1990.

[Kil92] J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 723–732, 1992.

[KLS93] S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. In *Proceedings of the 2nd Israel Symposium on Theory and Computing Systems, ISTCS*, pages 250–260. IEEE Computer Society Press, 1993.

[KMR93] D. Karger, R. Motwani, and G.D.S. Ramkumar. On approximating the longest path in a graph. In *Proceedings of Workshop on Algorithms and Data Structures*, pages 421–430. LNCS (Springer-Verlag), v. 709, 1993.

[KMS94] D. Karger, R. Motwani, and M. Sudan. Graph coloring through semi-definite programming. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, 1994.

[KMSV94] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. Computational versus syntactic views of approximability. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 819–830, 1994.

[Knu74] D.E. Knuth. *The Art of Computer Programming, vol. 2.* Addison-Wesley, Reading, Mass., 1974.

[Lev73] L. Levin. Universal'nyĭe perebornyĭe zadachi (universal search problems : in Russian). *Problemy Peredachi Informatsii*, 9(3):265–266, 1973.

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

[Lip89] R. Lipton. Efficient checking of computations. In *Proceedings of 6th STACS*, 1989.

[LLL82] A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:513–534, 1982.

[LLS90] J. Lagarias, H.W. Lenstra, and C.P. Schnorr. Korkine-Zolotarev bases and successive minima of a lattice and its reciprocal lattice. *Combinatorica*, 10:333–348, 1990.

[LO85] J.C. Lagarias and A.M. Odlyzko. Solving low-density subset-sum problems. *Journal of the ACM*, 32:229–246, 1985.

[Lov86] L. Lovász. *Algorithmic theory of numbers, graphs and convexity.* NSF-CBMS Reg. Conference Series. SIAM, 1986.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujam graphs. *Combinatorica*, 8:261–277, 1988.

[LR88] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 422–431, 1988.

[LS91] D. Lapidot and A. Shamir. Fully parallelized multi prover protocols for NEXPTIME. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 13–18, 1991.

[Lun92] C. Lund. *The Power of Interaction.* MIT Press, Cambridge, Mass., 1992.

[LV89] N. Linial and U. Vazirani. Graph-products and chromatic number. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 124–128, 1989.

[LY93] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In *Proceedings of International Colloquium on Automata, Languages and Programming, ICALP*, pages 40–51, 1993.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.

[Mic94] S. Micali. CS (Computationally Sound) proofs. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 436–453, 1994.

[MP68] M. Minsky and S. Papert. Perceptrons, 1968.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of error-correcting codes.* North-Holland, Amsterdam, 1977.

[Pap77] C. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4:237–244, 1977.

[Pap83] C. Papadimitriou. Games against nature. In *Proc. 24th IEEE Symp. on Foundations of Computer Science*, pages 446–450, 1983.

[PS94] A. Polishchuk and D. Spielman. Nearly linear size holographic proofs. In *Proc. 26th ACM Symp. on Theory of Computing*, 1994.

[PY91] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[PY93a] C. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the v-c dimension. In *Proc. 9th IEEE Structure in Complexity Theory Conference*, 1993.

[PY93b] C. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[Raz94] R. Raz. A parallel repetition theorem. Manuscript, 1994.

[RS92] R. Rubinfeld and M. Sudan. Testing polynomial functions efficiently and over rational domains. In *Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 23–32, 1992.

[Rub90] R. Rubinfeld. *A mathematical theory of self-checking, self-testing and self-correcting Programs*. PhD thesis, U.C. Berkeley, 1990.

[Sch85] C.P. Schnorr. A hierarchy of polynomial-time basis reduction algorithms. In *Proceedings of Conference on Algorithms, Pécs (Hungary)*, pages 375–386. North-Holland, 1985.

[SG76] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23:555–565, 1976.

[Sha92] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992.

[She91] A. Shen. Multilinearity test made easy. Manuscript, 1991.

[Sud92] M. Sudan. *Efficient checking of polynomials and proofs and the hardness of approximation problems*. PhD thesis, U.C. Berkeley, 1992.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[vEB81] P. van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical Report 81–04, Math. Inst. Univ. Amsterdam, 1981.

[VV86] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[Yan79] M. Yannakakis. The effect of a connectivity requirement on the complexity of of maximum subgraph problems. *Journal of the ACM*, 26:618–630, 1979.

[Yan81] M. Yannakakis. Edge deletion problems. *SIAM Journal of Computing*, 10:77–89, 1981.

[Yan92] M. Yannakakis. On the approximation of maximum satisfiability. In *Proceedings of 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–9, 1992.

[Yao90] A.C.C. Yao. Coherent functions and program checkers. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 84–94, 1990.

[Zuc91] D. Zuckerman. Simulating BPP using a general weak random source. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 79–89, 1991.

[Zuc93] D. Zuckerman. NP-complete problems have a version that's hard to approximate. In *8th Structure in Complexity Theory Conf.*, pages 305–312, 1993.

Appendix A

Library of Useful Facts

We include proofs of some simple facts assumed in the thesis.

Fact A.1 (Cook-Levin Theorem; Stronger Form): *For any language $L \in NP$ there is a fixed constant c such that given any input size n we can in time poly(n) construct a 3SAT formula ψ in the variables $x_1, \ldots, x_n, y_1, \ldots, y_{n^c}$ such that the an input $b \in \{0,1\}^n$ is in L iff*

$$\exists y_1, \ldots, y_{n^c} : \psi(b_1, \ldots, b_n, y_1, \ldots, y_{n^c}) = TRUE$$

where $b_1, \ldots, b_n$ are the bits of b.

Proof: Follows from an easy modification of the proof of the standard version of the Cook-Levin theorem. □

The following fact often goes by the name "Markov's inequality."

Fact A.2 (Averaging Principle): *Suppose the average of a set of numbers from $[0,1]$ is α. Then (i) The fraction of them that are greater than $k\alpha$ is at most $1/k$. (ii) The fraction of them that are greater than $\sqrt{\alpha}$ is less than $\sqrt{\alpha}$.*

Proof: (i) For, if not, then the average is more than $1/k \cdot k\alpha = \alpha$. This is a contradiction.

Part (ii) follows by using $k = 1/\sqrt{\alpha}$. □

In the rest of this section, F denotes the field GF(q). Some lemmas require q to be lowerbounded by a function of some other parameters.

Fact A.3: *For every set of k (point, value) pairs $\{(a_i, b_i) : 1 \le i \le k\}$, (where $a_i, b_i \in F$ and no point a_i is repeated in the list) there is a unique polynomial $p(x)$ of degree $k-1$ such that*

$$p(a_i) = b_i.$$

Proof: Let

$$L_i(x) = \prod_{j \neq i} \frac{(x - a_j)}{a_i - a_j}$$

be the polynomial that is 1 at a_i and zero at all a_j for $j \neq i$. Then the desired polynomial p is given by

$$p(x) = \sum_{i \leq k} b_i L_i(x).$$

Uniqueness is easy to verify. □

Fact A.4 (Schwartz): *An m-variate polynomial of degree d is 0 at no more than d/q fraction of points in F^m, where $q = |F|$.*

Proof: Proved by induction on m. Truth for $m = 1$ is clear, since a univariate degree d polynomial has at most d roots.

A degree-d polynomial $p(x_1, \ldots, x_m)$ has a representation as

$$\sum_{i=0}^{k} x_1^i \cdot p_i(x_2, \ldots, x_m) \tag{A.1}$$

where $k \leq d$ and each $p_i(x_2, \ldots, x_m)$ is a nonzero $(m-1)$-variate polynomial of degree at most $d - i$.

By the inductive hypothesis, for at least $(1 - (d-k)/q)$ fraction of values of $(x_2, \ldots, x_m)$, $p_k(x_2, \ldots, x_m) \neq 0$. For any such value of $(x_2, \ldots, x_m)$ the expression in Equation A.1 is a degree k polynomial in x_1, and so is zero for at most k values of x_1.

Hence the fraction of non-zeroes of p is at least $(1 - (d-k)/q)(1 - k/q) \geq (1 - d/q)$. □

Fact A.5: *Let $2d < q$ and f be an m-variate polynomial of degree at most d. If its restriction on d/q fraction of lines is $(1/2 + d/q)$-close to a univariate degree k polynomial, where $k < d$, then the degree of f is exactly k.*

Proof: A line is specified as $\{(u_1, \ldots, u_m) + t \cdot (v_1, \ldots, v_m) : t \in \mathrm{F}\}$ for some $u_1, \ldots, u_m$, $v_1, \ldots, v_m \in \mathrm{F}$. The restriction of f on such a line is given by

$$\sum_{i=0}^{d} t^i p_i(u_1, \ldots, u_m, v_1, \ldots, v_m)$$

where each p_i is a polynomial in $u_1, \ldots, u_m, v_1, \ldots, v_m$ of total degree at most d. In fact, a little thought shows that p_d is a function only of $v_1, \ldots, v_m$ and is exactly the sum of those terms in f whose total degree is d.

On any line where f's restriction is $(1/2+d/q)$-close to a univariate polynomial of degree k, $p_d, p_{d-1}, \ldots, p_k$ are 0. The hypothesis says that this happens for d/q fraction of the lines. Hence p_d must be identically zero. Thus f has no terms of total degree d. Repeating this argument shows that it has no terms of total degree more than k. □

Fact A.6: *Suppose $2d < q$ and $f : F^m \to F$ is a function whose restriction on every line is described by a univariate degree-d polynomial. Then $f \in F_d[x_1, \ldots, x_m]$.*

Proof: By induction on m. The case $m = 1$ is trivial.

Let $m > 1$. Let $a_0, \ldots, a_d$ be distinct points in F. According to the inductive hypothesis, the restriction of f on the $m-1$-dimensional subspace $\{(a_i, x_2, \ldots, x_m) : x_2, \ldots, x_m \in \mathrm{F}\}$ is described by an $(m-1)$-variate polynomial of degree d. Let f_i denote this polynomial.

Let L_i be the univariate polynomial that is 1 at a_i and 0 at all a_j for $j \neq i$. Consider the m-variate polynomial g defined as

$$\sum_{i=0}^{d} L_i(x_1) f_i(x_2, \ldots, x_m).$$

The restriction of g on any line that is parallel to the x_1-axis – that is, a line of the form $\{(t, b_2, \ldots, b_m) : t \in \mathrm{F}\}$ – is a degree d polynomial in x_1. This univariate polynomial describes f when $t = a_0, a_1, \ldots, a_d$. Hence it must be the univariate polynomial that describes f on the entire line.

Since the set of lines parallel to the x_1-axis intersects all points in F^m, we conclude that $g = f$. In particular, f is a polynomial of degree at most $2d$. But on every line it is described by a degree d univariate polynomial. So Fact A.5 implies that f has degree d. □

Fact A.7: *Let $A = (a_{ij})$ be an $n \times n$ matrix, where the entries a_{ij} are considered as variables. Then the determinant of A is a polynomial of degree n in the a_{ij}'s.*

Proof: Follows from inspection from the expression for the determinant.

$$\det(A) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \cdot \prod_{i \leq n} a_{i\sigma(i)},$$

where S_n is the set of all permutations of $\{1, \ldots, n\}$. □

The following fact is used in Section 5.1. The reader may choose to read it by mentally substituting "$\mathrm{F}[y]$, the set of polynomials over field F in a formal variable y" in place of R.

Fact A.8 (Cramer's Rule): *Let A be an $m \times n$ matrix whose entries come from an integral domain R, and $m > n$. Let $A \cdot x = 0$ be a system of m equations in n variables (note: it is an overdetermined homogeneous system).*

1. *The system has a non-trivial solution iff all $n \times n$ submatrices of A have determinant 0.*

2. *If the system does have a solution, then it has one of the type $(x_1 = u_1, \ldots, x_n = u_n)$ where each u_i is a sum of determinants of submatrices of A.*

Proof: Normally we would solve such equations by identifying the largest nonsingular submatrix of A, say B, fixing the variables that are not in B, and multiplying both sides by the inverse B^{-1} of B. Since the system is homogeneous, it is easily seen that we can also multiply both sides any matrix that is a scaling of B^{-1}. In this case we multiply by $det(B) \cdot B^{-1}$. But each entry of $\det(B) \cdot B^{-1}$ is itself a determinant of submatrices of B. Hence the claim is proved. $\square$

The following lemma uses terms defined in Definition 4.15 and 4.13. It says that the "average" curve in $P(< x_1, \ldots, x_k >)$ hits every subset of F^m of size $\mu \cdot |\mathrm{F}|^m$ in about $\mu \cdot |\mathrm{F}|$ points. Recall that when we say "set of points on a curve", we are actually talking about a multiset (for example, the sequence $x_1, \ldots, x_k$ below could have repetitions).

Lemma A.9 (Well-distribution Lemma for Curves): *Let $x_1, \ldots, x_k \in F^m$ be points (not all x_i's are the same) and $S \subseteq F^m$ be any set. Then the average of $|C \cap S|$, among all curves $C \in P(< x_1, \ldots, x_k >)$, is $\frac{|S|}{|F|^m}$.*

Proof: Let $\mathcal{C}$ be the set of curves of degree k whose first k points are $\{x_1, \ldots, x_k\}$. Consider the following enumeration of elements of $\mathcal{C}$.

> For every $x \in \mathrm{F}^m$ and j such that $k < j \leq |\mathrm{F}|$ the curve in $\mathcal{C}$ whose jth point is x.

This counts each curve in $\mathcal{C}$ and each $x \in F^m$ exactly $|\mathrm{F}| - k$ times.

Hence the average fraction of points from S on a curve in $\mathcal{C}$ is $\frac{|S|}{|\mathrm{F}|^m}$. $\square$

Fact A.10 (Geometry of a plane): *The lines and points of the plane F^2 have the following properties. (i) the number of lines is $q(q+1)$. (ii) Let two lines be* parallel *to each other if they do not intersect. For any fixed line, there are $q-1$ other lines parallel to it. (iii) Every line intersects q^2+1 other lines.*

Proof: Every two points determine a unique line. Every line has q points. Hence the the number of lines is $\binom{q^2}{2}/\binom{q}{2} = q(q+1)$. Thus (i) is proved.

The set of lines that are parallel to a given fixed line are mutually disjoint. Each has q points. Hence their number is at most $(q^2-q)/q = q-1$. It is easily seen that the number is exactly $q-1$. Thus (ii) is proved.

Finally, a line intersects every line it is not parallel to. The number of such lines, using (i) and (ii), is $q(q+1)-(q-1)$, which is q^2+1. Hence (iii) is proved.

□

Index