Computationally Efficient Error-Correcting Codes and Holographic Proofs

by

Daniel Alan Spielman

B.A., Mathematics and Computer Science
Yale University (1992)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1995

Signature of Author____
Department of Mathematics
May 5, 1995

Certified by____
Michael Sipser
Professor of Mathematics
Thesis Supervisor

Accepted by____
Richard Stanley, Chairman
Applied Mathematics Committee

Accepted by____
David Vogan, Chairman
Departmental Graduate Committee

Computationally Efficient Error-Correcting Codes and Holographic Proofs

by

Daniel Alan Spielman

Submitted to the Department of Mathematics
on May 5, 1995,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract

We present computationally efficient error-correcting codes and holographic proofs. Our error-correcting codes are asymptotically good and can be encoded and decoded in linear time. Our construction of holographic proofs provide, for every proof of any theorem, a slightly larger "holographic" proof whose accuracy can be probabilistically checked by an algorithm that only reads a constant number of the bits of the holographic proof and runs in poly-logarithmic time (such proofs have also been called "transparent proofs" and "probabilistically checkable proofs"). We explain how these constructions are related and how improvements of these constructions should result in a strengthening of this relationship.

For every constant r such that $0 < r < 1$, we construct an infinite family of systematic linear block error-correcting codes that have an encoding circuit with a linear number of wires. There is a constant $\epsilon > 0$ and a linear-time decoding algorithm for these codes that maps every word of relative distance at most ϵ from a codeword to that codeword. The encoding circuits have logarithmic depth. The decoding algorithm can be implemented as a circuit with $O(n \log n)$ wires and logarithmic depth. These constructions make use of explicit constructions of expander graphs and superconcentrators.

Our constructions of holographic proofs improve on the theorem $PCP(\log n, 1) = NP$, proved by Arora, Lund, Motwani, Sudan, and Szegedy, by providing, for every $\epsilon > 0$, constant-query checkable proofs of size $O(n^{1+\epsilon})$. That is, we design a probabilistic poly-logarithmic time proof checking algorithm that takes two inputs: a theorem candidate and a proof candidate. After reading a constant number of bits from each input, the proof checker decides whether to accept or reject its inputs. For every rigorous proof of length n of any theorem, there is an easily computable holographic proof of that theorem of size $O(n^{1+\epsilon})$ such that, with probability one, the proof checker will accept the holographic proof and an encoding of the theorem. Conversely, if the proof checker accepts a theorem candidate and a proof candidate with probability greater than one-half, then the theorem candidate is close to a unique encoding of a true theorem and the proof candidate constitutes a proof of that theorem.

Thesis Supervisor: Michael Sipser
Title: Professor of Mathematics

Credits

Most of the material in this thesis has appeared or will appear in conference proceedings. The material in Chapter 2 is joint work with Michael Sipser and appeared in [SS94]. The material in Chapter 4 appeared in [PS94] and is joint work with Alexander Polishchuk. I would like to thank Alexander Shen for putting me in contact with Polishchuk. The constructions in Chapter 3 will appear in [Spi95]. I thank both Michael Sipser and Alexander Polishchuk for these collaborations.

I would also like to thank the many people who have made useful suggestions to me in this research. Among them, I would especially like to thank Noga Alon, Oded Goldreich, Brendan Hassett, Marcos Kiwi, Carsten Lund, Nick Reingold, Michael Sipser, Madhu Sudan, and Shang-Hua Teng.

My thesis committee consisted of Michel Goemans, Shafi Goldwasser, and Michael Sipser.

Acknowledgments

I have benefited greatly from:

a special grade school education at The Philadelphia School, where we loved to learn without the motivation of grades, and we all cried at the end of the year;

the efforts of inspiring teachers—Carter Fussell, David Harbater, Serge Lang, Richard Davis, Jeffrey Macy, and Richard Beigel; I am fortunate to have encountered so many;

educational opportunities and support including the Young Scholars Program at the University of Pennsylvania, a Research Experience for Undergraduates from the National Science Foundation, a Fellowship from the Hertz Foundation, and three summers at AT&T Bell Laboratories;

the good advice and good listening of my advisor, Michael Sipser;

valuable advice, direction, and encouragement from Joan Feigenbaum;

my good friends Meredith Wright, Dave DuTot, Maarten Hoek, Marcos Kiwi, Jon Kleinberg, Shang-Hua and Suzie Teng, and The Crew; true friends are those who take pride in your achievements;

a close and loving family that gives me strength and makes me happy.

To Mom, Dad,

and Darren

Table of Contents

Chapter 1

Introduction

Mathematical studies of proofs and error-correcting codes played an important role in the development of theoretical computer science. In this dissertation, we develop ideas from theoretical computer science and apply them to the construction of error-correcting codes and proofs. Our goal is to construct error-correcting codes that can be encoded and decoded and proofs that can be verified as efficiently as possible. The error-correcting codes that we build are the first known asymptotically good family of error-correcting codes that can be encoded and decoded in linear time. Our construction of holographic proof systems enables one to transform any rigorous proof system into one whose proofs, while only slightly longer, can be probabilistically checked by the examination of only a constant number of their bits. We conclude by explaining how these constructions of error-correcting codes and holographic proofs are related.

1.1 Error-correcting codes

Error-correcting codes were introduced to deal with a fundamental problem in communication: when a message is sent from one place to another, it is often distorted along the way. An error-correcting code provides a systematic way of adding information to a message so that even if part of the message is corrupted in transmission, the receiver can nevertheless figure out what the sender intended to transmit. Naturally, the probability that the receiver can recover the original message decreases as the amount of distortion increases. Similarly, the amount of distortion that the receiver can tolerate increases as more redundant information is added to the transmitted message.

In his seminal 1948 paper, *A Mathematical Theory of Communication*, Shannon [Sha48] proved tight bounds on the amount of redundancy needed to tolerate a given amount of corruption in discrete communication channels. Shannon modeled the communication problem as a situation in which one is trying to send information from a source to a destination over a channel that occasionally becomes corrupted by noise (See Figure 1-1). Shannon described a coding scheme in which the transmitter breaks the original message into pieces, adds some redundant information to each piece to form a *codeword*, and then sends the codewords. The codewords are chosen so that if the received signal does not differ too much from the sent signal, the receiver will be able to remove the distortion from the received signal, figure out which codeword was sent, and pass the corrected message on to the destination. Shannon defined a notion called the *capacity* of a channel, which decreases as the amount of noise on the channel increases, and demonstrated that it is impossible to reliably send information from the source to the destination at a rate that exceeds the capacity of the channel. Thus, the capacity of the channel determines how much redundant information the transmitter needs to add to each piece of a message. Remarkably, Shannon was able to demonstrate that there is a coding scheme that enables one to reliably send information from the source to the destination at any rate lower than the capacity. Moreover, using such a coding scheme, one can arbitrarily reduce the probability that a message will fail to reach the destination without adding any extra redundant information: one need merely

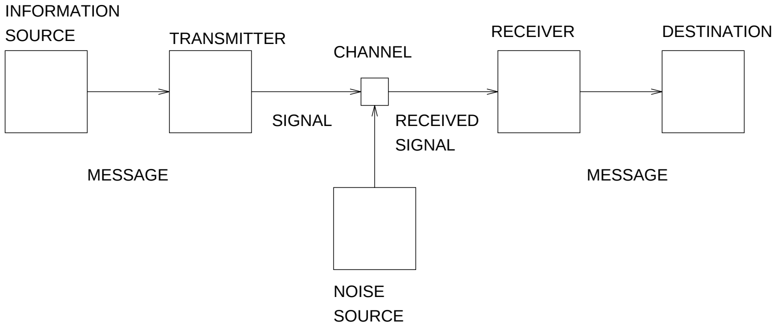

Figure 1-1: Shannon's schematic of a communication system.

increase the size of the pieces into which the message is broken.

But, there was a catch. While Shannon demonstrated that it is possible to encode longer and longer pieces of messages, he did not present a efficient means for doing so. Thus, the study of error-correcting codes was born in an attempt to find coding schemes that approximate the performance promised by Shannon and to find efficient implementations of these coding schemes. In this thesis, we present the first such coding scheme in which the computational effort needed to encode and decode corrupted codewords is strictly proportional to the length of the codewords. This enables us to arbitrarily reduce the probability of error in communication without decreasing the rate of transmission or increasing the computational work required.

1.2 The purpose of proofs

When we construct error-correcting codes, we are concerned with the transmission of a message, but we ignore its content. In the second part of this thesis, we explore how a receiver can become convinced that a message has a particular content while only reading a small portion of the message.[1] We formalize this by requiring that the message be a demonstration of some fact, and we consider the content of the message to be the truth of that fact. That is, we treat the message as a proof.

A proof is the means by which one mathematician demonstrates the truth of an assertion to another. Ideally, it should be much easier for the other to become convinced of the truth of the assertion by reading a proof of it than by attempting to derive the veracity of the assertion from scratch. Thus, we view a proof as a labor-saving device: by providing proofs, we provide others with the fruits of our labor, but spare them its pains[2]. By including extra details and background material, we make our proofs accessible to a broader audience. If we supply sufficient detail, then one almost completely ignorant of mathematics should be able to check the veracity of our claims.

In principle, any mathematical claim and accompanying proof can be expressed using

[1] Some may try to do this by reading only the introduction of the message, but one cannot be sure that the body will fulfill the promises made in the introduction.

[2] We hope that this dissertation achieves this goal.

a few axioms and a simple calculus. The verifier of such a proof need only be able to check that certain simple string operations are performed correctly and that strings have been faithfully copied from one part of the proof to another. By making the proofs slightly longer, it is possible to construct formats for proofs in which the verifier only needs to check that a collection of pre-specified conditions are met, each of which only involves a constant number of bits of the proof. At this point, it seems that we have made the task of the verifier as simple as is conceivably possible. But, by allowing for a little uncertainty, we can make it even simpler.

The complexity class non-deterministic polynomial time, usually called *NP*, roughly captures the set of facts that have proofs whose size is polynomial in the length of their statement. Similarly, the class non-deterministic exponential time, *NEXP*, captures the set of facts that have proofs whose size is exponential in the size of the statement of the fact. In their proof that $NEXP = MIP$, Babai, Fortnow, and Lund [BFL91] demonstrated that it is possible to *probabilistically* verify the truth of one of these exponentially long proofs while only reading a polynomial number of the bits of the proof. Without reading the whole proof, the verifier cannot be certain that it is true. But, by reading a small portion of the proof, the verifier can be very confident that it is true. That is, the verifier will always accept a correct proof of a true statement; but, if the verifier examines a purported proof of a false statement, then the verifier will reject the purported proof with high probability.

The following year, Babai, Fortnow, Levin, and Szegedy [BFLS91] explained how similar techniques could be combined with some new ideas to construct *transparent proofs* of any mathematical statement. Their transparent proof of a statement was only slightly longer than the formal proof from which it was derived, but it could be probabilistically verified in time poly-logarithmic (very small) in the size of the proof and the degree of confidence desired. In a related series of papers by Feige, Goldwasser, Lovász, Safra, and Szegedy [FGL$^+$91], Arora and Safra [AS92b], and Arora, Lund, Motwani, Sudan, and Szegedy [ALM$^+$92], proofs were created that could be probabilistically verified by examining only a constant number of randomly chosen bits of the proof.[3] However, the size of

[3] These types of proofs have gone by the names *transparent proofs*, *probabilistically checkable proofs*, and *holographic proofs*. We prefer the name *holographic proofs*, introduced by Levin, because, as in a hologram,

these proofs was at least quadratic in the size of the original proofs.

We construct proofs that combine the advantages of being nearly linear in the size of the original proofs and having verifiers that run in poly-logarithmic time and read only a constant number of bits of the proof.

Note that we do not advocate the use of such proof systems by practicing mathematicians. A mathematician is rarely satisfied with merely knowing that a fact is true. The excitement of mathematics lies in understanding why. While one might obtain such an understanding by reading a holographic proof, it would be much simpler to read a proof written in plain language designed for easy comprehension. We study holographic proofs both because we want to find out how little work one need do to verify a fact and because, in the process of studying this fundamental question, we obtain results that have important implications in other areas.

Among the techniques used to construct and analyze holographic proofs, we would like to point out the importance of those derived from the study of: checkable, self-testable/self-correctable, and random-self-reducible functions [BK89, RS92, BLR90, Rub90, GLR$^+$91, GS92, Lip91, BF90, Sud92, She91]; techniques for reducing our dependence on randomness [IZ89, Zuc91]; interactive proofs [Bab85, BM88, GMR89, GMR85, BoGKW88, FRS88, LFKN90, Sha90, FL92, LS91, BFL91]; and error-correcting codes [GS92, BW, BFLS91]. Through many of these works one finds a common algebraic thread inspired by the work of Schwartz [Sch80]. The crux of our construction of holographic proofs is a purely algebraic statement—Theorem 4.2.19.

The main application of constructions of holographic proofs has been to prove the hardness of finding approximate solutions to certain optimization problems. Notable papers in this direction include [PY91, FGL$^+$91, AS92b, ALM$^+$92, LY94, Zuc93, BGLR93, F9̈4, BS94, ABSS93]. Holographic proofs have also been applied to problems in many areas of complexity theory [CFLS93, CFLS94, Kil92, Kil94, Mic94, KLR$^+$94]. While not properly an application of holographic proof technology, our constructions of error-correcting codes were

each bit of information in a holographic proof is reflected in the entire structure. Some authors reserve the term *probabilistically checkable* for proof systems in which the proof checker reads the statement to be proved and use *transparent* or *holographic* to describe systems in which the statement of the theorem to be proved is encoded so that the proof checker only needs to read a constant number of its bits.

inspired by these proofs. They were born of an attempt to find an alternative construction of holographic proofs, and we hope that they one day aid in this effort.

1.3 Structure of this thesis

We begin in Section 1.4 with an introduction to the field of error-correcting codes from the perspective of this complexity theorist. In Section 1.5, we describe some of what is known about the complexity of error-correcting codes. The goal of this introduction is to provide a context for the results in Chapters 2 and 3.

Chapters 2 and 3 are devoted to our construction of linear-time encodable and decodable error-correcting codes. In Chapter 2, we describe a construction of codes that can be decoded in linear time, but for which we only know quadratic-time encoding algorithms. These codes can also be decoded in logarithmic time by a linear number of processors. In this chapter, we introduce many of the techniques that we will use in Chapter 3. In particular, we present a relation between expander graphs and error-correcting codes. Along the way, we survey some of what is known about expander graphs. As we feel that these codes might be useful for coding on write-once media, we conclude the chapter by presenting some thoughts on how one might go about implementing these codes.

In Chapter 3, we present our construction of linear-time encodable and decodable error-correcting codes. These codes can also be encoded and decoded in logarithmic time with a linear number of processors. We call these codes *superconcentrator codes* because their encoding circuits bear a strong resemblance to superconcentrators. As part of our construction, we develop a type of code that we call an *error-reducing code*. An error-reducing code enables one to quickly remove most of the errors from a corrupted codeword, but it need not allow full error-correction. We construct superconcentrator codes by carefully piecing together appropriately chosen error-reducing codes. Again, we conclude this chapter with some thoughts on how one might implement these codes and on how they could be improved.

Chapter 4 is devoted to our construction of nearly linear size holographic proofs. This Chapter is somewhat more involved than Chapters 2 and 3, but it should be intelligible to the reader with a reasonable background in theoretical computer science. We begin by describing holographic proofs and by defining some of the types of error-correcting codes that they are related to. A potentially useful type of error-correcting code derived from holographic proofs is a *checkable code*. Checkable codes have associated randomized algorithms that, after examining only a constant number of randomly chosen bits of a received word, can make a good estimation of the probability that a decoder will successfully decode the word. One could use such an algorithm to decide whether to request the retransmission of a word even before the decoder has worked on it. In Section 4.2, we develop the algebraic machinery that we will need to construct our holographic proofs. We show that certain polynomial codes are somewhat checkable and verifiable. These codes are used in Section 4.3 to construct our basic holographic proof system. We apply this system to itself recursively to construct our efficient holographic proofs. In the final sections, we present a few variations of our construction, each more powerful, but more complicated, than the previous.

In the last chapter of this thesis, we explain some of the connections between our constructions of error-correcting codes and holographic proofs. We explain the similarities between expander codes and the checkable codes derived from holographic proofs. We also discuss whether checkable codes are a necessary component of holographic proofs. Our conclusion is that the problems of constructing more efficient checkable codes, holographic proofs, and expander codes are strongly linked, and could even have the same solution.

1.4 An introduction to error-correcting codes

The purpose of this section is to provide the reader with a convenient reference for the basic definitions from the field of error-correcting codes that we will use in this thesis. Those who understand the phrase "asymptotically good family of systematic linear error-correcting codes" can probably skip this section.

Intuitively, a good error-correcting code is a large set of words such that each pair differs in many places. Let Σ be a finite alphabet with q letters. A *code of length n over Σ* is a subset of Σ^n. Throughout most of Chapters 2 and 3, we will discuss codes over the alphabet $\{0, 1\}$, which are called *binary* codes. Unless otherwise stated, all the codes that we discuss should be assumed to be binary.

Two important parameters of a code are its *rate* and *minimum distance*. The *rate* of a code $\mathcal{C}$ is $(\log_q |\mathcal{C}|)/n$. The rate indicates how much information is contained, on average, in each code symbol. The *minimum distance* of a code $\mathcal{C}$ is

$$\min_{\substack{x,y \in \mathcal{C} \\ x \neq y}} d(x, y),$$

where $d(x, y)$ is the Hamming-distance between two words (*i.e.*, the number of places in which they differ). We will usually talk about the *relative minimum distance* of a code, $d(x, y)/n$.

Since we are interested in the asymptotic performance of codes, we define a *family of error-correcting codes* to be an infinite sequence of error-correcting codes that contains at most one code of any length. If $\{\mathcal{C}_i\}$ is a family of error-correcting codes such that, for all i, the rate of $\mathcal{C}_i$ is greater than r, then we say that the family has rate at least r. Similarly, if the relative minimum distance of each code in the family is at least δ, then we say that the relative minimum distance of the family is at least δ. An infinite family of codes over a fixed alphabet is called *asymptotically good* if there exist positive constants r and δ such that the family has rate and relative minimum distance at least r and δ respectively.[4] A central problem of coding theory has been to find explicit constructions of asymptotically

[4] We will occasionally say *good* code when we mean asymptotically good code.

good families of error-correcting codes with as large rate and relative minimum distance as possible.

The preceding definitions are standard. We will now make some less standard definitions that will help us discuss the complexity of encoding and decoding error-correcting codes.

An *encoding function* for a code $\mathcal{C}$ of rate r is a bijection

$$f : \Sigma^{rn} \rightarrow \mathcal{C}.$$

We say that an encoding function is *systematic* if there exist indices $i_1, \ldots, i_{rn}$ such that for all $\vec{x} = (x_1, \ldots, x_{rn}) \in \Sigma^{rn}$,

$$(x_1, \ldots, x_{rn}) = (f(\vec{x})_{i_1}, \ldots, f(\vec{x})_{i_{rn}}).$$

That is, the message is embedded in the codeword. If a code has a systematic encoding function, then we say that the code is *systematic*. We can think of a systematic code as being divided into rn "message symbols" and $(1-r)n$ "check symbols". The check symbols are uniquely determined by the message symbols, and we can view the message symbols as containing the information content of the codeword (in a binary code, we will call these message bits and check bits).

An *error-correcting function* for a code $\mathcal{C}$ is a function

$$g : \Sigma^n \rightarrow \mathcal{C} \cup \{?\}$$

such that $g(x) = x$, for all $x \in \mathcal{C}$. We say that an error-correcting function for $\mathcal{C}$ can correct m errors if for all $x \in \mathcal{C}$ and all $y \in \Sigma^n$ such that $d(x,y) \leq m$, we have $g(y) = x$. We allow an error-correcting function to return the value "?" so that it can describe the output of an algorithm that could not find an element of $\mathcal{C}$ close to its input. We use the verb *decode* loosely to indicate the process of correcting a constant fraction of errors. When we say a "decoding algorithm", we mean an algorithm that can correct some constant fraction of errors. From the perspective of a complexity theorist, the fraction of errors that can be efficiently corrected in a family of error-correcting codes is much more interesting than the actual minimum distance of those codes.

1.4.1 Linear codes

The codes that we construct will be *linear codes.* A *linear code* is a code whose alphabet is a field and whose codewords form a vector space over this field. There are two natural ways to represent a linear code: either by listing a basis of the space and defining the code to be all linear combinations of those basis vectors, or by listing a basis of the dual space. A matrix whose rows are a basis of the dual space is called a *check matrix* of the code.

In a binary code, each check bit is just a sum modulo 2 of a subset of the message bits. Some elementary facts that we will use about linear binary codes are:

- The zero vector is always a codeword.
- They are systematic. To see this, row reduce the $(1-r)n \times n$ check matrix until it contains $(1-r)n$ columns that contain exactly one 1. These columns correspond to the check bits, and the remaining rn columns correspond to the message bits. Observe that row-reducing the check matrix does not change the code that it defines. It is now clear that for any setting of the message bits, there is a unique setting of the check bits so that the resulting vector is orthogonal[5] to every row in the row-reduced check matrix.
- They can be encoded using $O(n^2)$ work: there is a $rn \times (1-r)n$ matrix such that the check bits can be computed from the message bits by multiplying the vector of rn message bits by this matrix. (This is the matrix that appears in the columns corresponding to the message bits described in the previous item.)
- The minimum distance of a linear code is equal to the minimum weight of a non-zero codeword (the weight of a codeword w is $d(0,w)$; note that $d(v,w) = d(0, w-v)$).

[5] We say that two vectors are orthogonal if their inner product is zero. Over a finite field, this loses its geometric significance: a vector can be orthogonal to itself!

1.4.2 Asymptotic bounds

Good linear codes are easy to construct. A randomly chosen parity check matrix defines a good code with exponentially high probability.

Theorem 1.4.1 Let $v_1, \ldots, v_{\alpha n}$ be vectors chosen uniformly at random from $GF(2)^n$. Let $\mathcal{C}$ be the code consisting of the length n vectors over $GF(2)$ that are orthogonal to all of $v_1, \ldots, v_{\alpha n}$. $\mathcal{C}$ has rate at least $1 - \alpha$. With high probability, $\mathcal{C}$ has relative minimum distance at least ϵ, for $\epsilon < 1/2$ and $\alpha > H(\epsilon)$, where $H(\cdot)$ is the binary entropy function ($i.e.$ $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$).

Proof: The probability that any non-zero word is orthogonal to each of $v_1, \ldots, v_{\alpha n}$ is $2^{-\alpha n}$. Thus, the probability that some non-zero vector of weight at most ϵn is orthogonal to each of $v_1, \ldots, v_{\alpha n}$ is at most $\sum_{i=1}^{\epsilon n} \binom{n}{i} 2^{-\alpha n}$. One can use Stirling's formula to show that, for fixed ϵ, $\log_2 \binom{n}{\epsilon n} = nH(\epsilon) + O(\log n)$. Thus, the sum approaches zero if $\alpha > H(\epsilon)$. ■

Codes with rate r and minimum relative distance ϵ for $r \geq 1 - H(\epsilon)$ are said to meet the *Gilbert-Varshamov bound.* While these random linear codes may be encoded using quadratic work, we know of no efficient algorithm for decoding them.

An easy upper bound on the performance of a code is the sphere-packing bound:

Theorem 1.4.2 Let $\mathcal{C}_n$ be an infinite sequence of codes with relative minimum distance at least δ. Then

$$\limsup_{n \to \infty} \text{rate}(\mathcal{C}_n) \leq 1 - H(\delta/2).$$

Proof: If the code has minimum distance δn, then there are disjoint balls of radius $\delta n/2$ around each codeword. These balls account for at least $2^{rn}\binom{n}{\delta n/2}$ words; but, there are only 2^n words of length n. ■

Better upper bounds than the sphere-packing bound are known. The Elias bound, which held the record for a long time, has a fairly simple proof.

Theorem 1.4.3 [Elias] Let $\mathcal{C}_n$ be an infinite sequence of codes with relative minimum distance at least δ. Then

$$\limsup_{n \to \infty} \text{rate}(\mathcal{C}_n) \leq 1 - H\left(\frac{1}{2} - \frac{1}{2}\sqrt{1 - 2\delta}\right).$$

Even better, but much more complicated to prove, is the McEliece-Rodemich-Rumsey-Welch upper bound:

Theorem 1.4.4 [McEliece-Rodemich-Rumsey-Welch] Let $\mathcal{C}_n$ be an infinite sequence of codes with relative minimum distance at least δ. Then

$$\limsup_{n\to\infty} \ \mathsf{rate}(\mathcal{C}_n) \leq H\left(\frac{1}{2} - \sqrt{\delta(1-\delta)}\right).$$

Proofs of both of these can be found in [MS77].

1.5 The complexity of coding

Many of the foundations for the analysis of the complexity of algorithms were laid by Shannon in his 1949 paper *The synthesis of two-terminal switching circuits* [Sha49]. Suggestions for how to analyze the complexity issues special to error-correcting codes appeared in the work of Savage [Sav69, Sav71] and Bassalygo, Zyablov, and Pinsker [BZP77]. For more general studies of the complexity of algorithms, we point the reader to [AHU74] and [CLR90].

From our perspective, the relevant questions are: how hard is it to encode a family of error-correcting codes, and how much time does it take to correct a constant fraction of errors? We are not as interested in the minimum distance of a code as we are in the number of errors that we can efficiently correct. We measure encoding and decoding efficiency by the time of a RAM algorithm or the size and depth of a boolean circuit that performs the operations. Bassalygo, Zyablov, and Pinsker point out that we should also examine the complexity of building the encoding and decoding programs or circuits. While we do not discuss this in detail, it will be clear that there are efficient polynomial-time algorithms for constructing our encoding and decoding programs and circuits.

Initially, the only algorithms known for decoding error-correcting codes were exponential in complexity: enumerate all codewords and select one closest to the received word. For linear codes, one could do slightly better by computing which parity checks were violated and searching for the smallest pattern of errors that would violate exactly that set of parity checks. There were numerous improvements on these approaches. We will not attempt to survey the accomplishments of coding theory; we will just mention the most efficient coding algorithms known prior to our work: Using efficient implementations of the Finite Fourier Transform, Justesen [Jus76] and Sarwate [Sar77] have shown that certain Reed-Solomon and Goppa codes can be encoded in $O(n \log n)$ time and decoded in time $O\left(n \log^2 n\right)$. While these codes are not necessarily asymptotically good, one can compose them with good codes to obtain asymptotically good codes with similar encoding and decoding times. Moreover, these algorithms are easily parallelized.

Codes that have more efficient algorithms for one of these operations have suffered in the other. Gelfand, Dobrushin, and Pinsker [GDP73] presented randomized constructions of asymptotically good codes that could be encoded in linear time. However, they did not suggest algorithms for decoding their codes, and we suspect that a polynomial-time algorithm would be difficult to find.

Zyablov and Pinsker [ZP76] showed that it is possible to decode Gallager's randomly chosen low-density parity-check codes [Gal63] in logarithmic time with a linear number of processors. These codes are essentially the same as those we present in Section 2.3. We are not aware of any algorithm for encoding these codes that uses less than $O(n^2)$ work. Kuznetsov [Kuz73] used these codes to construct fault-tolerant memories. Pippenger has pointed out that Kuznetsov's proof of the correctness of these memories can serve as a proof of correctness of the parallel decoding algorithm that we present in Section 2.3.3. By analyzing these codes in terms of the expansion properties of the graphs by which they are defined, we are able to provide a much simpler proof of the correctness of the parallel decoding algorithm, prove for the first time the correctness of the natural sequential decoding algorithm presented in Section 2.3.1, and obtain the first explicit constructions of asymptotically good low-density parity-check codes.

CHAPTER 2

Expander codes

In this chapter, we explain a way of using expander graphs to construct asymptotically good linear error-correcting codes. These codes can be decoded in linear sequential time or parallel logarithmic time with a linear number of processors. The best encoding algorithms that we know for these codes are the $O(n^2)$ time algorithms that can be used for all linear codes. These codes fall into the category of low-density parity-check codes introduced by Gallager [Gal63]. The construction that we present in Section 2.5 is the first known explicit construction of an asymptotically good family of low-density codes.

We begin by explaining what expander graphs are and prove that good expander graphs exist. In Section 2.2, we define expander codes precisely and prove that they can be asymptotically good. In Section 2.3, we show how error-correcting codes derived from very good expander graphs can be efficiently decoded. The construction that we present in this section is similar to Gallager's construction. In Section 2.3.1, we provide the first proof of correctness for the natural sequential algorithm for decoding these codes. In Section 2.3.2, we demonstrate that this algorithm only works on codes derived from expander graphs. Thus, Gallager's codes work precisely when the graphs from which they are derived are expanders. Unfortunately, we are not aware of deterministic constructions of expander graphs with the level of expansion needed for this first construction.

In Section 2.4, we survey some of what is known about explicit constructions of expander graphs. We use these constructions in Section 2.5 to produce explicit constructions of expander codes that can be decoded efficiently.

We conclude this chapter with a discussion of how one might want to implement these codes and a presentation of the results of experiments which demonstrate the good performance of these codes.

2.1 Introduction to expander graphs

An expander graph is a graph in which every set of vertices has a large number of neighbors. It is a perhaps surprising but nonetheless well known fact that expander graphs that expand by a constant factor, but which have only a linear number of edges, do exist. In fact, a simple randomized process will produce such a graph.

Let $G = (V, E)$ be a graph on n vertices. To describe the expansion properties of G, we say *every set of size at most m expands by a factor of c* if, for all sets $S \subset V$,

$$|S| \leq m \qquad \Rightarrow \qquad \Big| \{y : \exists x \in S \text{ such that } (x, y) \in E\} \Big| \geq c\,|S|\,.$$

One can show that for all $\epsilon > 0$ there exists a $\delta > 0$ such that, for sufficiently large n, a random d-regular graph will probably expand by a factor of $d - 1 - \epsilon$ on sets of size δn. We cannot hope to find graphs that expand by a factor greater than $d - 1$ because it is easy to find sets that have this level of expansion: the vertices of a cycle in a graph does the trick (a graph of degree greater than two has cycles of logarithmic size). Ideally, we should describe the expansion of a graph by presenting a function that gives the expansion factor for each size of set.

In our constructions, we will make use of unbalanced bipartite expander graphs. That is, the vertices of the graph will be divided into two sets such that there are no edges between vertices in the same set. We will call such a graph (d, c)*-regular* if all the nodes in one set have degree d and all the nodes in the other have degree c. By counting edges, we find that the number of d-regular vertices must differ from the number of c-regular vertices by a factor of c/d. We will only consider the expansion of sets of vertices contained within one side of the graph. In Section 2.1.1, we will show that if $c > d$ and one chooses a (d, c)-regular graph at random, then it will have expansion approaching $d - 1$ from the large side to the small side and expansion approaching $c - 1$ from the small side to the large side.

We wish to point out that Alon, Bruck, Naor, Naor, and Roth [ABN$^+$92] used expander graphs in a different way to construct asymptotically good families of error-correcting codes that lie above the Zyablov bound [MS77]. Also, Alon and Roichman [AR94] use error-correcting codes to construct expander graphs.

2.1.1 The expansion of random graphs

In this section, we will prove upper and lower bounds on the expansion factors achieved by random graphs that become tight as the degrees of the graphs become large.

We first prove a simple upper bound on the expansion any graph can achieve.

Theorem 2.1.1 Let B be a bipartite graph between n d-regular vertices and $\frac{d}{c}n$ c-regular vertices. For all $0 < \alpha < 1$, there exists a set of αn d-regular vertices with at most

$$n\frac{d}{c}\left(1 - (1 - \alpha)^c\right) + O(1) \text{ neighbors.}$$

Proof: Choose a set X of αn d-regular vertices uniformly at random. Now, consider the probability that a given c-regular vertex is not a neighbor of the set of d-regular vertices. Each neighbor of the c-regular vertex is in the set X with probability α. Thus, the probability that the c-regular vertex is not a neighbor of X is

$$\prod_{i=0}^{c-1} \frac{n - \alpha n - i}{n - i},$$

which tends to $(1 - \alpha)^c$ as n grows large. This implies that the expected number of non-neighbors tends to $n\frac{d}{c}(1 - \alpha)^c$. ■

This simple upper bound becomes tight as d grows large.

How to choose a random (d, c)-regular graph: To choose a random (d, c)-regular bipartite graph, we first choose a random matching between dn "left" nodes and dn "right" nodes. We collapse consecutive sets of d left nodes to form the n d-regular vertices, and we collapse consecutive sets of c right nodes to form the $\frac{d}{c}n$ c-regular vertices. It is possible that this graph will have multiedges that should be thrown away, but this does not hurt the lower bound on the expansion of this graph that we will prove.

To make our language consistent with the rest of this chapter, we will call the d-regular vertices "variables" and the c-regular vertices "constraints".

Theorem 2.1.2 Let B be a randomly chosen (d, c)-regular bipartite graph between n variables and $\frac{d}{c}n$ constraints. Then, for all $0 < \alpha < 1$, with exponentially high probability all sets of αn variables in B have at least

$$n\left(\frac{d}{c}(1-(1-\alpha)^c) - \sqrt{2d\alpha H(\alpha)/\log_2 e}\right)$$

neighbors, where $H(\cdot)$ is the binary entropy function.

Proof: First, we fix a set of αn variables, V, and estimate the probability that V's set of neighbors is small. The probability that a given constraint is a neighbor of V is at least $1-(1-\alpha)^c$. Thus, the expected number of neighbors of V is at least $n\frac{d}{c}(1-(1-\alpha)^c)$. Noga Alon suggested that we form a martingale (See [AS92a]) to bound the probability that the size of the set of neighbors deviates from this expectation.

Each node in V will have d outgoing edges. We will consider the process in which the destinations of these edges are revealed one at a time. We will let X_i be the random variable equal to the expected size of the set of neighbors of V given that the first i edges leaving V have been revealed. $X_1, \ldots, X_{d\alpha n}$ form a martingale such that

$$|X_{i+1} - X_i| \leq 1,$$

for all $0 \leq i < d\alpha n$. Thus, by Azuma's Inequality (See [AS92a]),

$$\text{Prob}[E[X_{d\alpha n}] - X_{d\alpha n} > \lambda\sqrt{d\alpha n}] < e^{-\lambda^2/2}.$$

But, $E[X_{d\alpha n}]$ is just the expected number of neighbors of V. Moreover, $X_{d\alpha n}$ is the expected size of the set of neighbors of V given that *all* edges leaving V have been revealed, which is exactly the size of the set of neighbors of V.

Since there are only $\binom{n}{\alpha n}$ choices for the set V, it suffices to choose λ so that

$$\binom{n}{\alpha n} e^{-\lambda^2/2} << 1.$$

By Stirling's formula, this holds for large n if λ satisfies

$$nH(\alpha)/\log_2 e < \lambda^2/2 \qquad \Rightarrow \qquad \sqrt{2nH(\alpha)/\log_2 e} < \lambda.$$

In general, if a graph has good expansion on a certain size set, then it will have similar expansion on smaller sets. However, this is not always true. What we can say is that the probability that sets of a certain size have a given expansion factor is a unimodal function. Theorem 2.1.2 shows that the probability of failure is exponentially small for large sets. However, for sets of constant size the probability of failure will be only polynomially small. One can probably show that whenever the probability that large sets have a certain expansion factor is exponentially small, all smaller sets will have the same expansion factor with high probability.

2.2 Expander codes

To build an expander code, we begin with an unbalanced bipartite expander graph. Say that the graph is (d, c)-regular between sets of vertices of size n and $\frac{d}{c}n$, and that $c > d$. We will identify each of the n nodes on the large side of the graph with one of the bits in a code of length n. We will usually refer to these n bits as *variables*. Each of the $\frac{d}{c}n$ vertices on the small side of the graph will be associated with a constraint. Each constraint will restrict only those variables that are neighbors of the vertex identified with the constraint (See Figure 2-1). These will be called the "variables in the constraint". A constraint will require that the variables it restricts form a codeword in some linear code of length c. Because each constraint we impose upon the variables is linear, the expander codes we construct will be linear as well. It is convenient to let all the constraints be the same.

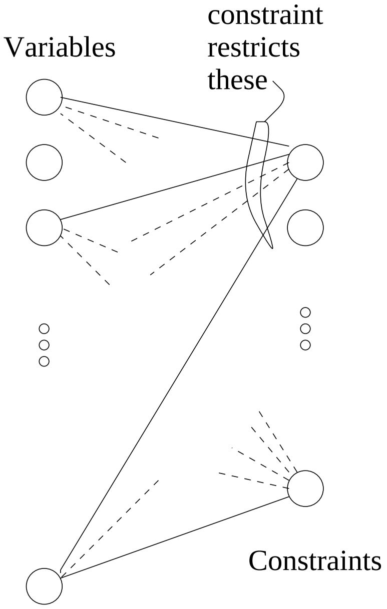

Figure 2-1: A constraint restricts the variables that are its neighbors.

Definition 2.2.1 Let B be a bipartite graph between n variables and $\frac{d}{c}n$ constraints that is d-regular on the variables and c-regular on the constraints. Let $\mathcal{S}$ be a code of block-length c. A constraint is *satisfied* by a setting of the variables if the variables in that constraint form a codeword of $\mathcal{S}$. The expander code $\mathcal{C}(B, \mathcal{S})$ consists of the settings of the variables that satisfy every constraint.

If the expander graph is a sufficiently good expander and if the constraints are identified with sufficiently good codes, then the resulting expander code will be a good code.

Theorem 2.2.2 Let B be a bipartite graph between n variables and $\frac{d}{c}n$ constraints that is d-regular on the variables and c-regular on the constraints. Let $\mathcal{S}$ be a code of block-length c, rate r, and relative minimum distance ϵ. If B expands by a factor of more than $\frac{d}{c\epsilon}$ on all sets of size at most αn, then $\mathcal{C}(B, \mathcal{S})$ has rate at least $dr - (d-1)$ and relative minimum distance at least α.

Proof: To obtain the bound on the rate of the code, we will count the number of linear restrictions imposed by the constraints. Each constraint induces $(1-r)c$ linear restrictions. Thus, there are a total of

$$n\frac{d}{c}(1-r)c = dn(1-r)$$

linear restrictions, which implies that there are at least $n(dr - (d-1))$ degrees of freedom.

To prove the bound on the minimum distance, we will show that there can be no non-zero codeword of weight less than αn. Let w be a non-zero word of weight at most αn and let V be the set of variables that are 1 in this word. There are $d|V|$ edges leaving the variables in V. The expansion property of the graph implies that these edges will enter more than $\frac{d}{c\epsilon}|V|$ constraints. Thus, the average number of edges per constraint will be less than $c\epsilon$, so there must be some constraint that is a neighbor of V, but which has a number of neighbors in V that is less than the minimum distance of $\mathcal{S}$. This implies that w cannot induce a codeword of S in that constraint; so, w cannot be a codeword in $\mathcal{C}(B, S)$. ■

Remark 2.2.3 A construction of codes defined by identifying the nodes on one side of a bipartite graph with the bits of the code and identifying the nodes on the other side with subcodes first appeared in the work of Tanner [Tan81]. Following Gallager's lead, Tanner analyzed the performance of his codes by examining the girth of the bipartite graph. Margulis [Mar73] also used high-girth graphs to construct error-correcting codes. Unfortunately, it seems that analysis resting on high-girth is insufficient to demonstrate that families of codes are asymptotically good.

2.3 A simple example

A simple example of expander codes is obtained by letting B be a graph with expansion greater than $\frac{d}{2}$ on sets of size at most αn, and letting $\mathcal{S}$ be the code consisting of words of even weight. The parity-check matrix of the resulting code, $\mathcal{C}(B, \mathcal{S})$, is just the adjacency matrix of B. The code $\mathcal{S}$ has rate $\frac{c-1}{c}$ and minimum relative distance $\frac{2}{c}$, so $\mathcal{C}(B, \mathcal{S})$ has rate $1 - \frac{d}{c}$ and minimum distance at least αn.

To obtain a code that we can decode efficiently, we will need even greater expansion. With greater expansion, small sets of corrupt variables will induce non-codewords in many constraints. By examining these "unsatisfied" constraints, we will be able to determine which variables are corrupted. In Sections 2.3.1 and 2.3.3, we will explain how to decode these simple expander codes.

Unfortunately, we do not know of explicit constructions of expander graphs with expansion greater than $\frac{d}{2}$. Thus, in order to construct these simple codes, we must use the randomized construction of expanders explained in Section 2.1.

2.3.1 Sequential decoding

There is a natural algorithm for decoding these simple expander codes. We say that a constraint is "satisfied" by a word w if the sum of the values that w assigns to the variables in the constraint is even; otherwise, we call the constraint "unsatisfied". Consider what happens when we flip[1] a variable that is in more unsatisfied than satisfied constraints. The unsatisfied constraints containing the variable become satisfied, and *vice versa*. Thus, we have decreased the total number of unsatisfied constraints. The idea behind the sequential decoding algorithm is to keep doing this until no unsatisfied constraints remain, in which case we have a codeword. Theorem 2.3.1 says that if the graph used to define the code is a good expander and if not too many variables of a codeword are corrupted, then this algorithm will succeed.

Sequential expander code decoding algorithm:

- If there is a variable that is in more unsatisfied than satisfied constraints, then flip the value of that variable.
- Repeat until no such variables remain.

It is easy to implement this algorithm so that it runs in linear time (assuming that pointer references have unit cost). In Figure 2-2, we present one such way of implementing this algorithm. We assume that the graph has been provided to the algorithm as a graph of pointers in which each constraint points to the variables it contains, and each variable points to the constraints in which it appears. The implementation runs in two phases: a set-up phase that requires linear time, and then a loop that takes constant time per iteration. During the set-up phase, the variables are partitioned into lists by the number of unsatisfied constraints in which they appear. During normal iteration of the loop, a variable

[1] If the variable was 0, make it 1. If it was 1, make it 0.

```
Set-up phase
  For each constraint, compute the parity of the sum of the variables it contains. (The
  algorithm should have a list of the constraints.)
  Initialize lists $L_0, \ldots, L_d$.
  For each variable, count the number of unsatisfied constraints in which it appears.
  If this number is $i$, then put the variable in list $L_i$.
Loop
  Until lists $L_{\lceil d/2 \rceil}, \ldots, L_d$ are empty do:
    Find the greatest $i$ such that $L_i$ is not empty
    Choose a variable $v$ from list $L_i$
    Flip the value of variable $v$
    For each constraint $c$ that contains variable $v$
      Update the status of constraint $c$
      For each variable $w$ in constraint $c$
        Recompute the number of unsatisfied constraints in which $w$ appears. Move it
        to the appropriate list.
  If all variables are in list $L_0$, the output the values of the variables. Otherwise,
  report "failed to decode".
```

Figure 2-2: An implementation of sequential expander code decoding algorithm.

that appears in the greatest number of unsatisfied constraints is flipped; the status of each constraint that contains that variable is updated; and each variable that appears in each of those constraints is moved to the list that reflects its new number of unsatisfied constraints. If, at some point, there is no variable in more unsatisfied than satisfied constraints, the implementation leaves the loop and checks whether it has successfully decoded its input. If all the variables are in the list L_0, then there are no unsatisfied constraints and the implementation will output a codeword. We will show that the loop is executed at most a linear number of times.

Theorem 2.3.1 Let B be a bipartite graph between n variables of degree d and $\frac{d}{c}n$ constraints of degree c such that all sets X of at most αn variables have at least $(\frac{3}{4} + \epsilon)d|X|$ neighbors, for some $\epsilon > 0$. Let $\mathcal{C}(B)$ be the code consisting of those settings of the variables that cause every constraint to have parity zero. Then the sequential decoding algorithm will correct up to an $\alpha/2$ fraction of errors while executing the decoding loop at most $d\alpha n/2$ times. Moreover, the algorithm runs in linear time on all inputs, regardless of whether or not B is a good expander.

Proof: We will say that the decoding algorithm is in state (v, u) if v variables are corrupted and u constraints are unsatisfied. We view u as a potential associated with v. Our goal is to demonstrate that the potential will eventually reach zero. To do this, we will show that if the decoding algorithm begins with a word of weight at most $\alpha n/2$, then, at every step, there will be some variable with more unsatisfied neighbors than satisfied neighbors.

First, we consider what happens when the algorithm is in a state (v, u) with $v < \alpha n$. Let s be the number of satisfied neighbors of the corrupted variables. By the expansion of the graph, we know that

$$u + s \geq \left(\frac{3}{4} + \epsilon\right) dv.$$

Because each satisfied neighbor of the corrupted variables must share at least two edges with the corrupted variables, and each unsatisfied neighbor must have at least one, we know that

$$dv \geq u + 2s.$$

By combining these two inequalities, we obtain

$$s \leq \left(\frac{1}{4} - \epsilon\right) dv \qquad \text{and} \qquad u \geq \left(\frac{1}{2} + 2\epsilon\right) dv. \tag{2.1}$$

Since each unsatisfied constraint must share at least one edge with a corrupted variable, and since there are only dv edges leaving the corrupted variables, we see that at least a $(\frac{1}{2} + 2\epsilon)$ fraction of the edges leaving the corrupted variables must enter unsatisfied constraints. This implies that there must be some corrupted variable such that a $(\frac{1}{2} + 2\epsilon)$ fraction of its neighbors are unsatisfied. Of course, this does not mean that the decoding algorithm will decide to flip a corrupted variable.

However, it does mean that the only way that the algorithm could fail to decode is if it flips so many uncorrupt variables that v becomes greater than αn. Assume by way of contradiction that this happens. Then there must be some time at which v equals αn. At this time, equation (2.1) tells us that $u > \frac{d}{2}\alpha n$. This leads to a contradiction because u is initially at most $\frac{d}{2}\alpha n$ and can only decrease during the execution of the algorithm.

This would imply that the algorithm was in a state $(\alpha n, u)$, where $u < \frac{d}{2}\alpha n$, because u was initially at most $\frac{d}{2}\alpha n$. But, this would contradict the analysis above.

To see that the implementation of the sequential decoding algorithm runs in linear time, first observe that the degree of every variable and constraint is constant; so, the set-up phase requires linear time. Moreover, every time a variable is flipped by the algorithm, the number of unsatisfied constraints decreases. Thus, the loop cannot be executed a number of times greater than the number of constraints. Because the degree of each variable and constraint is constant, each iteration requires only constant time. ■

We note that it is possible to improve the constants in this analysis by taking into account the fact that after each decoding step, the number of corrupted variables actually decreases by at least $2\frac{u}{v} - d$.

2.3.2 Necessity of Expansion

We will now show that the sequential decoding algorithm works only if the graph B is an expander graph.

Theorem 2.3.2 Let B be a bipartite graph between n variables of degree d and $\frac{d}{c}n$ constraints of degree c such that the sequential expander code decoding algorithm successfully decodes all sets of at most αn errors in the code $\mathcal{C}(B)$. Then, all sets of αn variables must have at least

$$\alpha n \left(1 + \frac{2\frac{d-1}{2c}}{3 + \frac{d-1}{2c}}\right)$$

neighbors in B.

Proof: We will first deal with the case in which d is even. In this case, every time a variable is flipped, the number of unsatisfied constraints decreases by at least 2. Consider the performance of the decoding algorithm on a word of weight αn. Because the algorithm stops when the number of unsatisfied constraints reaches zero, the algorithm must decrease the number of unsatisfied constraints by at least $2\alpha n$ as it corrects the αn corrupted variables. Thus, every word of weight αn must cause at least $2\alpha n$ constraints to be unsatisfied, so every set of αn variables must have at least $2\alpha n$ neighbors. Because we assume that $c > d$,

$$2 > 1 + \frac{2\frac{d-1}{2c}}{3 + \frac{d-1}{2c}},$$

and we are done with the case in which d is even.

When d is odd, we can only guarantee that the number of unsatisfied constraints will decrease by 1 at each iteration. This means that every set of αn variables must induce at least αn unsatisfied constraints. Alone, this is insufficient to demonstrate expansion by a factor greater than 1. However, let us consider what must happen for the algorithm to be in a state in which αn variables are corrupted, but there is no variable that the decoding algorithm can flip that will cause the number of unsatisfied constraints to decrease by more than 1. This means that each corrupted variable has at least $\frac{d-1}{2}$ of its edges in satisfied constraints. Because each satisfied constraint can have at most c incoming edges, this implies that there must be at least $\alpha n \frac{d-1}{2c}$ satisfied neighbors of the αn variables. Thus, the set of αn variables must have at least $\alpha n(1 + \frac{d-1}{2c})$ neighbors.

On the other hand, if the algorithm decreases the number of unsatisfied constraints by more than 1, then it must decrease the number by at least 3. For some word of weight αn, assume that the algorithm flips $\beta \alpha n$ variables before it flips a variable that decreases the number of unsatisfied constraints by only 1. The original set of αn variables must have had at least

$$3\beta\alpha n + (1-\beta)\alpha n$$

neighbors. On the other hand, once the algorithm flips a variable that causes the number of unsatisfied constraints to decrease by 1, we can apply the bound of the previous paragraph to see that the variables must have at least

$$(1-\beta)\alpha n \left(1 + \frac{d-1}{2c}\right)$$

neighbors. We note that this bound is strictly decreasing in β, while the previous bound is strictly increasing in β, so the lower bound that we can obtain on the expansion occurs when β is chosen so that

$$\begin{aligned} 3\beta\alpha n + (1-\beta)\alpha n &= (1+\tfrac{d-1}{2c})\,(1-\beta)\alpha n & \Rightarrow \\ 1+2\beta &= (1-\beta)\,(1+\tfrac{d-1}{2c}) & \Rightarrow \\ \beta\,(3+\tfrac{d-1}{2c}) &= \tfrac{d-1}{2c} & \Rightarrow \\ \beta &= \tfrac{\frac{d-1}{2c}}{3+\frac{d-1}{2c}}. & \end{aligned}$$

When we plug β back in, we find that the set of αn variables must have at least

$$\begin{aligned} \alpha n \left(3\left(\frac{\frac{d-1}{2c}}{3+\frac{d-1}{2c}}\right) + \left(1 - \frac{\frac{d-1}{2c}}{3+\frac{d-1}{2c}}\right)\right) &= \alpha n \left(\frac{3+3\frac{d-1}{2c}}{3+\frac{d-1}{2c}}\right) \\ &= \alpha n \left(1 + \frac{2\frac{d-1}{2c}}{3+\frac{d-1}{2c}}\right) \end{aligned}$$

neighbors. ■

2.3.3 Parallel decoding

The sequential decoding algorithm has a natural parallel analogue: in parallel, flip each variable that appears in more unsatisfied than satisfied constraints. We will see that this algorithm can also correct a constant fraction of errors if the code is derived from a sufficiently good expander graph.

Parallel expander code decoding algorithm:

- In parallel, flip each variable that is in more unsatisfied than satisfied constraints.
- Repeat until no such variables remain.

Theorem 2.3.3 Let B be a bipartite graph between n variables of degree d and $\frac{d}{c}n$ constraints of degree c such that all sets X of at most $\alpha_0 n$ variables have more than $(\frac{3}{4}+\epsilon)d|X|$ neighbors, for some $\epsilon > 0$. Let $\mathcal{C}(B)$ be the code consisting of those settings of the variables that cause every constraint to have parity zero. Then, $\mathcal{C}(B)$ has rate at least $(1-\frac{d}{c})$ and the parallel expander code decoding algorithm will correct any $\alpha < \frac{\alpha_0(1+4\epsilon)}{2}$ fraction of errors after $\log_{1/(1-2\epsilon)}(\alpha n)$ decoding rounds, where each round requires constant time.

Proof: Assume that the algorithm is presented with a word of weight at most αn, where $\alpha < \frac{\alpha_0(1+4\epsilon)}{2}$. We will refer to the variables that are 1 as the *corrupted variables*, and we will let S denote this set of variables. We will show that after one decoding round, the algorithm will produce a word that has at most $(1-2\epsilon)\alpha n$ corrupted variables.

To this end, we will examine the sizes of F, the set of corrupted variables that fail to flip in one decoding round, and C, the set of variables that were originally uncorrupt, but which become corrupt after one decoding round. After one decoding round, the set of corrupted variables will be $C \cup F$. Define $v = |S|$, and set ϕ, γ, and δ so that $|F| = \phi v$, $|C| = \gamma v$, and $|N(S)| = \delta d v$. By expansion, $\delta > \frac{3}{4} + \epsilon$. To prove the theorem, we will show that

$$\phi + \gamma < \frac{\frac{1}{4}}{\frac{1}{4} + \epsilon} < 1 - 2\epsilon.$$

We will first show that

$$\phi < 4 - 4\delta.$$

We can bound the number of neighbors of S by observing that each variable in F must share at least half of its neighbors with other corrupted variables. Thus, each variable in F can account for at most $\frac{3}{4}d$ neighbors, and, of course, each variable in $S \setminus F$ can account for most d neighbors, which implies

$$\begin{aligned} \delta d v &\leq \frac{3}{4} d\phi v + d(1-\phi)v \qquad \Rightarrow \\ \phi &\leq 4 - 4\delta. \end{aligned}$$

We now show that

$$\gamma < \frac{\delta - (\frac{3}{4} + \epsilon)}{\frac{1}{4} + \epsilon}.$$

Assume by way of contradiction that this is false. Let C' be a subset of C of size $\frac{\delta - (\frac{3}{4}+\epsilon)}{\frac{1}{4}+\epsilon} v$. Each variable in C' must have at least $\frac{d}{2}$ edges that land in constraints that are neighbors of S. Thus, the total number of neighbors of $C' \cup S$ is at most

$$\frac{d}{2}|C'| + \delta d v.$$

But, because the set $C' \cup S$ has size at most

$$\begin{aligned} &\left(1 + \frac{\delta - (\frac{3}{4} + \epsilon)}{\frac{1}{4} + \epsilon}\right)\left(\frac{1 + 4\epsilon}{2}\right)\alpha_0 n \\ &\leq \left(\frac{\frac{1}{2}}{\frac{1}{4} + \epsilon}\right)\left(\frac{1 + 4\epsilon}{2}\right)\alpha_0 n = \alpha_0 n, \end{aligned}$$

it must have expansion greater than $d(\frac{3}{4} + \epsilon)$, which implies that

$$\begin{aligned} \frac{d}{2}|C'| + \delta d v &= \frac{d}{2}\left(\frac{\delta - (\frac{3}{4} + \epsilon)}{\frac{1}{4} + \epsilon}\right) v + \delta d v \\ &> \left(\frac{3}{4} + \epsilon\right) d \left(1 + \frac{\delta - (\frac{3}{4} + \epsilon)}{\frac{1}{4} + \epsilon}\right) v. \end{aligned}$$

After simplifying this inequality, we find that it is a contradiction. Thus, we find

$$\begin{aligned} \phi + \gamma &< 4 - 4\delta + \frac{\delta - (\frac{3}{4} + \epsilon)}{\frac{1}{4} + \epsilon} \\ &= \frac{1 - (\frac{3}{4} + \epsilon) + 4\epsilon - 4\epsilon\delta}{\frac{1}{4} + \epsilon} \\ &= \frac{\frac{1}{4} - \epsilon + 4\epsilon(1 - \delta)}{\frac{1}{4} + \epsilon} \\ &< \frac{\frac{1}{4} - \epsilon + 4\epsilon(\frac{1}{4})}{\frac{1}{4} + \epsilon} \\ &= \frac{\frac{1}{4}}{\frac{1}{4} + \epsilon}. \end{aligned}$$

■

We note that this algorithm can be implemented by a circuit of size $O(n \log n)$ and depth $O(\log n)$.

2.4 Explicit constructions of expander graphs

In order to create explicit constructions of expander codes, we will need to make use of explicit constructions of expander graphs. In this section, we will survey some of what is known about explicit constructions of expander graphs. The only facts in this section that are necessary for the results of this Chapter are Lemma 2.4.6, Lemma 2.4.14, and Theorem 2.4.7. In Chapter 3, we will also make use of Proposition 2.4.19.

The expansion properties of a graph are strongly related to its eigenvalues. We recall their definition:

Let G be a graph on n vertices. The adjacency matrix of G is the n-by-n matrix, A, with entries $(a_{i,j})$ such that

$$a_{i,j} = \begin{cases} 1, \text{ if } (i,j) \text{ is an edge of } G, \text{ and} \\ 0, \text{otherwise.} \end{cases}$$

When we say "the eigenvalues of G", we mean the eigenvalues of this matrix.

Remark 2.4.1 Some researchers normalize the entries in the adjacency matrix so that its eigenvalues have absolute value at most 1. It is also common to study the Laplacian of a graph—the matrix that has the degree of vertex i as the (i,i)-th entry, -1 as the (i,j)-th entry if i and j are adjacent, and 0 otherwise. This matrix has the advantage of being positive semi-definite.

We will now restrict our discussion to graphs that are regular of some degree d. Note that d will be an eigenvalue of these graphs, corresponding to the eigenvector that is 1 in each entry. It is fairly easy to show that the multiplicity of the eigenvalue d will be equal to the number of connected components of G. Henceforth, we will only consider connected graphs G. It is again fairly easy to show that $-d$ will be an eigenvalue if and only if the graph is bipartite. All other eigenvalues must lie between $-d$ and d.

A graph will be a good expander if all eigenvalues other than d have small absolute value.[2] For example, a large separation of eigenvalues implies that the number of edges between two sets of vertices is roughly the number expected if the sets were chosen at random:

Theorem 2.4.2 [Beigel-Margulis-Spielman] Let $G = (V, E)$ be a d-regular connected graph

[2] The analogous statement holds for bipartite graphs if all eigenvalues other than d and $-d$ are small, but we will not discuss that situation here.

on n vertices such that every eigenvalue other than d has absolute value at most λ. Let X and Y be subsets of V of sizes αn and βn respectively. Then

$$\left| |\{(x,y) \in X \times Y : (x,y) \in E\}| - d\alpha\beta n \right| \leq n\lambda\sqrt{(\alpha - \alpha^2)(\beta - \beta^2)}.$$

Proof: Let A be the adjacency matrix of the graph G, and let $\vec{x}$ and $\vec{y}$ be the characteristic vectors of the sets X and Y respectively (*i.e.* the vector that is 1 for each vertex in the set, and 0 otherwise).

We will compute $\langle A\vec{x}, \vec{y}\rangle$ in two ways, where $\langle \cdot, \cdot \rangle$ denotes the standard inner product of vectors. We first observe that this term counts the number of edges from vertices in X to vertices in Y. (If X and Y intersect, then each edge in their intersection is counted twice as it can be viewed as going from X to Y in two ways.) So,

$$\langle A\vec{x}, \vec{y}\rangle = |\{(x,y) \in X \times Y : (x,y) \in E\}|\,.$$

Let $\vec{v} = \vec{x} - \alpha\vec{1}$ and let $\vec{w} = \vec{y} - \beta\vec{1}$. A simple calculation reveals that $\vec{v}$ and $\vec{w}$ are perpendicular to $\vec{1}$. Because A is a symmetric matrix, we also know that $A\vec{v}$ is perpendicular to $\vec{1}$.

We compute

$$\begin{aligned}
\langle A\vec{x}, \vec{y}\rangle &= \left\langle A(\vec{v} + \alpha\vec{1}), (\vec{w} + \beta\vec{1})\right\rangle \\
&= \langle A\vec{v}, \vec{w}\rangle + \left\langle A\vec{v}, \beta\vec{1}\right\rangle + \left\langle \alpha A\vec{1}, \vec{w}\right\rangle + \left\langle \alpha A\vec{1}, \beta\vec{1}\right\rangle \\
&= \langle A\vec{v}, \vec{w}\rangle + \left\langle A\alpha\vec{1}, \beta\vec{1}\right\rangle \\
&= \langle A\vec{v}, \vec{w}\rangle + d\alpha\beta n
\end{aligned}$$

because $A\vec{v}$ is perpendicular to $\vec{1}$, and $\vec{w}$ is perpendicular to $A\vec{1}$. Recall that for two vectors $\vec{a}$ and $\vec{b}$, $\langle a, b\rangle \leq \sqrt{\langle a, a\rangle \langle b, b\rangle}$. We combine this with the observations that

$$\begin{aligned}
\langle \vec{v}, \vec{v}\rangle &= \alpha(1-\alpha)n, \\
\langle \vec{w}, \vec{w}\rangle &= \beta(1-\beta)n, \text{ and}
\end{aligned}$$

$\vec{v}$ is perpendicular to the eigenvector corresponding to d to obtain

$$\begin{aligned} |\langle A\vec{v}, \vec{w}\rangle| &\leq \lambda n\sqrt{(\alpha-\alpha^2)(\beta-\beta^2)} \qquad \Rightarrow \\ |\langle A\vec{x}, \vec{y}\rangle - dn\alpha\beta| &\leq \lambda n\sqrt{(\alpha-\alpha^2)(\beta-\beta^2)}. \end{aligned}$$ ■

We will use this result to derive Tanner's theorem [Tan84], which shows that if λ is small, then G is an expander.

Theorem 2.4.3 [Tanner] Let G be a d-regular graph on n vertices such that all eigenvalues other than d have absolute value at most λ. Then, for any subset X of the vertices of G, the neighbors of X, $N(X)$, must have size

$$|N(X)| \geq \frac{d^2|X|}{\lambda^2 + (d^2-\lambda^2)\frac{|X|}{n}}.$$

Proof: Let X have size αn. Let Y be the set of non-neighbors of X, and assume Y has size βn. Because there are no edges between X and Y, Theorem 2.4.2 implies that

$$\begin{aligned} d\alpha\beta n &\leq n\lambda\sqrt{(\alpha-\alpha^2)(\beta-\beta^2)} && \Rightarrow \\ d^2\alpha\beta &\leq \lambda^2(1-\alpha)(1-\beta) && \Rightarrow \\ \beta\left(\alpha(d^2-\lambda^2)+\lambda^2\right) &\leq \lambda^2(1-\alpha) && \Rightarrow \\ 1-\beta &\geq \tfrac{\alpha d^2}{\alpha(d^2-\lambda^2)+\lambda^2}. \end{aligned}$$

Remark 2.4.4 One can similarly show that a d-regular bipartite graph will be a good expander if every eigenvalue other than d and $-d$ is small. These results do not actually depend on the graph being d-regular. One can prove similar statements so long as there is a separation between the first and second largest eigenvalues.

Remark 2.4.5 Kahale [Kah93b] improves this theorem by showing that graphs with a similar eigenvalue separation have even better expansion.

The property of expander graphs that will be most useful to us in this dissertation is that small subsets of vertices in an expander have very small induced subgraphs:

Lemma 2.4.6 [Alon-Chung [AC88]] Let G be a d-regular graph on n vertices such that all eigenvalues other than d have absolute value at most λ. Let X be a subset of the vertices of G of size γn. Then, the number of edges contained in the subgraph induced by X in G is at most

$$\frac{n}{2}d\left(\gamma^2 + \frac{\lambda}{d}\gamma(1-\gamma)\right).$$

Proof: Follows from Theorem 2.4.2 by setting $Y = X$ and observing that we count every edge of the induced subgraph twice. ■

It is natural to ask how large a separation can exist between the eigenvalue d and the next-largest eigenvalue. Lubotzky, Phillips and Sarnak [LPS88] and, independently, Margulis [Mar88] provide an explicit construction of infinite families of graphs in which there is a very large separation.

Theorem 2.4.7 [Lubotzky-Phillips-Sarnak, Margulis] For every pair of primes p, q congruent to 1 modulo 4 such that p is a quadratic residue modulo q, there is an easily constructible $(p+1)$-regular graph with $q(q^2-1)/2$ vertices such that the second-largest eigenvalue of the graph is at most $2\sqrt{p}$.

We will hereafter refer to these graphs as LPS-M graphs.

Remark 2.4.8 LPS-M graphs are Cayley graphs of $PSL(2, \mathbf{Z}/q\mathbf{Z})$. One can find a representation of these graphs so that the names of the neighbors of a vertex can be computed in poly-logarithmic time from the name of that vertex.

Alon and Boppana [Alo86] have proved that the separation of eigenvalues achieved by LPS-M graphs is optimal:

Theorem 2.4.9 [Alon-Boppana] Let G_n be a sequence of k-regular graphs on n vertices and let λ_n be the second-largest eigenvalue of G_n. Then

$$\limsup_{n\to\infty} \lambda_n \geq 2\sqrt{k-1}.$$

Alon [Alo86] has proved that every expander graph must have some separation of eigenvalues.

Theorem 2.4.10 Let G be a d-regular graph on n vertices such that for all sets S of size at most $n/2$,

$$|N(S)| \geq (1+c)\,|S|\,.$$

Then, all eigenvalues of G but d have absolute value at most

$$d - \frac{c^2}{4+c^2}.$$

LPS-M graphs will suffice for our constructions in Section 2.5. However, a more general class of graphs will suffice as well. The following definition captures this class of graphs:

Definition 2.4.11 We will say that a family of graphs $\mathcal{G}$ is a *family of good expander graphs* if $\mathcal{G}$ contains graphs $G_{n,d}$ of n nodes and degree d so that

- For an infinite number of values of d, there exists an infinite number of graphs $G_{n,d} \in \mathcal{G}$, and
- for each of these values d, the second-largest eigenvalues of $G_{n,d}$ are bounded from above by constants λ_d such that $\lim_{d\to\infty} \lambda_d/d = 0$.

Pippenger [Pip93] points out that we can obtain a family of good expander graphs by exponentiating the expander graphs constructed by Gabber and Galil. Gabber and Galil [GG81] construct an infinite family of 5-regular expander graphs. By Theorem 2.4.10, all the eigenvalues of these graphs other than 5 must be bounded by some number $\lambda < 5$. Consider the graphs obtained by exponentiating the adjacency matrices of these graphs k times. They will have degree 5^k, and all eigenvalues but the largest will have absolute value at most λ^k. So, these yield a family of good expander graphs. These graphs will have multiple edges and self-loops, but this is irrelevant in our applications. Moreover, one could remove the multiple edges and self-loops without adversely affecting the quality of the expanders.

To construct unbalanced bipartite expander graphs, we will take the edge-vertex incidence graphs of ordinary expanders. From a d-regular graph G on n vertices, we will derive a $(2, d)$-regular graph with $dn/2$ vertices on one side and n vertices on the other.

Definition 2.4.12 Let G be a graph with edge set E and vertex set V. The *edge-vertex incidence graph* of G is the bipartite graph with vertex set $E \cup V$ and edge set

$$\{(e, v) \in E \times V : v \text{ is an endpoint of } e\}.$$

Our analyses of these graphs will follow from Lemma 2.4.6.

For some constructions, we will want graphs of higher degree. To construct these, we use a more general construction due to Ajtai, Komlós and Szemerédi [AKS87]:

Definition 2.4.13 Let G be a graph with edge set E and vertex set V. The k-path-vertex incidence graph of G is a bipartite graph between the set of paths of length k in G and the vertices of G in which a vertex of G is connected to each path on which it lies. (A path of length k is a sequence of vertices $v_1, \ldots, v_{k+1}$ such that $(v_i, v_{i+1}) \in E$ for each $1 \leq i \leq k$.)

We analyze the expansion properties of these graphs through the following fact proved by Kahale [Kah93a].

Lemma 2.4.14 [Kahale] Let $G_{n,d}$ be a d-regular graph on n nodes with second-largest eigenvalue bounded by λ. Let S be a subset of the vertices of $G_{n,d}$ of size γn. Then, the number of paths of length k contained in the subgraph induced by S in $G_{n,d}$ is at most

$$\gamma n d^k \left(\gamma + \lambda(1-\gamma)\right)^k.$$

A proof of this fact can also be found in [AFWZ]. The difference of a factor of two between Lemma 2.4.14 when $k = 1$ and Lemma 2.4.6 is due to the fact that in Lemma 2.4.14 each path is being counted twice: once for each endpoint.

2.4.1 Expander graphs of every size

It is natural to wonder how often the conditions of Theorem 2.4.7 are met. By the law of quadratic reciprocity, p is a quadratic residue modulo q if and only if q is a quadratic residue modulo p, for p and q primes congruent to 1 modulo 4. Thus, for a given p, we can bound the gaps between successive q's that satisfy the conditions of Theorem 2.4.7 by bounding the sizes of gaps between successive primes in arithmetic progressions. One can do this using the following theorem of Heilbronn [Hei33]:

Theorem 2.4.15 [Heilbronn] Let $\pi(x; k, l)$ denote the number of primes less than x and congruent to l modulo k. Then, there exists a constant $a < 1$ such that

$$\lim_{x \to \infty} \left(\pi(x + x^a; k, l) - \pi(x; k, l)\right) \frac{\ln x}{x^a} = \frac{1}{\phi(k)},$$

where l is relatively prime to k and $\phi(k)$ denotes the number of positive integers less than and relatively prime to k.

Definition 2.4.16 Let $\mathcal{G} = \{G_i\}_{i=1}^{\infty}$ be a sequence of graphs such that G_i has n_i vertices, and $n_{i+1} > n_i$. We say that $\mathcal{G}$ is *dense* if $n_{i+1} - n_i = o(n_i)$.

Theorem 2.4.15 implies that LPS-M graphs are dense. It is not difficult to see that the graphs obtained by exponentiating the expander graphs of Gabber and Galil are dense as well.

A dense family provides expander graphs of size close to almost every number; but, we will need good expanders of every sufficiently large number of vertices for our constructions in Chapter 3. We build such a family from a dense family by removing a few nodes from the expanders in such a way that they remain good expanders.

Definition 2.4.17 Let G be a graph. If S is a subset of the vertices of G such that no two vertices of S are neighbors or share a common neighbor, then we call S a *doubly independent* set of vertices.

If G is a d-regular graph with n vertices, then it is easy to see that it will have a doubly-independent set of at least $n/(d^2 + 1)$ vertices. Moreover, we can show that if a doubly-independent set of vertices is removed from a good expander graph, then it remains a good expander graph.

Proposition 2.4.18 Let G be a d-regular graph on n vertices such that all eigenvalues but the largest have absolute value at most λ. Let G' be a graph obtained from G by removing a doubly-independent set of vertices. Then, for any subset X of the vertices of G', the neighbors of X, $N(X)$, must have size

$$|N(X)| \geq |X| \left(\frac{d^2}{\lambda^2 + (d^2 - \lambda^2)\frac{|X|}{n}} - 1 \right).$$

Proof: We will show that the set of neighbors of X in G' can be smaller than its set of neighbors in G by at most one neighbor for each vertex in X. Let S denote the set of vertices that were removed from G to obtain G'. The only vertices that could be neighbors of X in G but not in G' are those that are in S. However, if there are two vertices in S that are both neighbors of one vertex in X, then S could not be a doubly independent set. Thus, the proposition follows from Theorem 2.4.3. ■

Proposition 2.4.19 Let G be a d-regular graph on n vertices such that all eigenvalues but the largest have absolute value at most λ. Let G' denote a graph that can be obtained from G by removing a doubly-independent set of vertices, S. Let X be a subset of the vertices of G' of size γn. Then, the number of edges contained in the subgraph induced by X in G is at most

$$\frac{n}{2} d \left(\gamma^2 + \frac{\lambda}{d} \gamma (1 - \gamma) \right).$$

Moreover, each vertex of G' has degree either d or $d - 1$.

Proof: Follows from Lemma 2.4.6 because the subgraph induced by X in G' is the same as the subgraph induced by X in G. ■

Remark 2.4.20 Not only can we find expander graphs with any number of vertices, but we can find expander graphs with any number of edges as well. We need merely choose a graph that has a few more edges than we desire, and remove a few of them. It is easy to remove edges so that each vertex still has degree d or $d-1$ and the graph has expansion properties similar to those obtained in Propositions 2.4.18 and 2.4.19.

For more information about expander graphs and algebraic graph theory, we recommend:

- Nabil Kahale. *Exander Graphs.* PhD thesis, M.I.T., September 1993.
- Norman Biggs. *Algebraic Graph Theory.* Cambridge University Press, New York, NY, second edition, 1993.
- Dragos M. Cvetkovic, Michael Doob, and Horst Sachs. *Spectra of graphs : theory and application.* Academic Press, New York, 1990.

2.5 Explicit constructions of expander codes

Let G be a d-regular graph in which every eigenvalue but d has absolute value at most λ. Let B be the edge-vertex incidence graph of G. We will construct codes of the form $\mathcal{C}(B, \mathcal{S})$, where $\mathcal{S}$ is itself a fairly good code of constant block length (recall Definition 2.2.1). The reason that we will be able to decode these codes is that when a constraint is unsatisfied, we know much more than just this bit of information. Usually, there is a codeword of $\mathcal{S}$ that is closest to the word described by the variables in the constraint. If this word is sufficiently close to the codeword, then the algorithm will flip those variables that must be changed in order to obtain that codeword.

Assume that $\mathcal{S}$ has minimum relative distance ϵ.

Parallel explicit expander code decoding round

- In parallel, for each constraint, if the setting of the variables in that constraint is within $\epsilon/4$ of a codeword, then send a "flip" message to every variable that needs to be flipped to obtain that codeword.
- In parallel, every variable that receives the message "flip", flips its value.

Remark 2.5.1 One parallel explicit expander code decoding round can be performed by $O(n)$ processors in constant time. Similarly, one could implement this as a linear-size circuit of constant depth.

Lemma 2.5.2 Let $\mathcal{S}$ be a linear code with rate r, minimum relative distance ϵ and block-length d. Let G be a d-regular graph such that all eigenvalues but d have absolute value at most λ. Let B be the edge-vertex incidence graph of G. Then $\mathcal{C}(B, \mathcal{S})$ has rate $2r - 1$ and minimum relative distance at least

$$\delta = \gamma^2 + (\gamma - \gamma^2)\frac{\lambda}{d}, \text{ where}$$

$$\gamma d + (1 - \gamma)\lambda \leq \epsilon d.$$

Moreover, if

$$\frac{\epsilon^2}{768} > \frac{\epsilon}{16} \cdot \frac{\lambda}{d},$$

then the iteration of $O(\log n)$ parallel explicit expander code decoding rounds will correct a $\frac{\epsilon^2}{64}$ fraction of errors. This algorithm can be simulated on a sequential machine in linear time.

Proof: Lemma 2.4.6 implies that B expands by a factor of at least

$$\frac{\gamma n}{\frac{nd}{2}\left(\gamma^2 + \frac{\lambda}{d}(\gamma - \gamma^2)\right)}$$

on sets of variables of size

$$\frac{nd}{2}\left(\gamma^2 + \frac{\lambda}{d}(\gamma - \gamma^2)\right).$$

Using this bound, the bounds on the rate and minimum distance of $\mathcal{C}(B_{n,d}, S)$ follow immediately from Theorem 2.2.2.

Let X be a set of at most $\alpha\frac{nd}{2}$ variables. We will show that if a decoding round is given a word that is 1 for all $v \in X$, and 0 elsewhere, then it will output a word whose weight is less by a constant factor, provided that α is not too big.

We will say that a constraint is *confused* if it sends a "flip" message to a variable that is not in X. In order for this to happen, the constraint must have at least $\frac{3}{4}\epsilon d$ variables of X as neighbors. Each variable of X can be a neighbor of two constraints, so there can be at most

$$\frac{\alpha\frac{nd}{2} \cdot 2}{\frac{3}{4}\epsilon d} = \frac{4\alpha n}{3\epsilon}$$

confused constraints. Each of these can send at most $\frac{\epsilon}{4}d$ "flip" signals, so at most

$$\frac{4\alpha n}{3\epsilon} \cdot \frac{\epsilon}{4} d = \frac{nd}{2} \cdot \frac{2\alpha}{3}$$

variables not in X will receive "flip" signals.

We will call a constraint *unhelpful* if it has more than $\frac{\epsilon}{4}$ neighbors in X. A variable in X will flip unless both of its neighbors are in unhelpful constraints. There are at most

$$\frac{\alpha \frac{d}{2} n \cdot 2}{\frac{1}{4}\epsilon d} = \frac{4\alpha n}{\epsilon}$$

unhelpful constraints, which by Lemma 2.4.6 can have at most

$$\frac{nd}{2}\left(\left(\frac{4\alpha}{\epsilon}\right)^2 + \left(\frac{4\alpha}{\epsilon}\right)\frac{\lambda}{d}\right)$$

edges among them. So, the weight of the output codeword must decrease by at least

$$\frac{nd}{2}\left(\alpha - \frac{2\alpha}{3} - \left(\frac{4\alpha}{\epsilon}\right)^2 - \frac{4\alpha}{\epsilon} \cdot \frac{\lambda}{d}\right)$$

The worst case is when α is maximized. If we set $\alpha = \frac{\epsilon^2}{64}$, then we obtain

$$\frac{nd}{2}\left(\frac{\epsilon^2}{768} - \frac{\epsilon}{16} \cdot \frac{\lambda}{d}\right).$$

Thus, the weight of the output codeword must decrease by a constant factor.

To simulate this algorithm in linear time, one should maintain a list of the unsatisfied constraints. When simulating any round after the first, the algorithm should only consider the variables that appear in those constraints. We have proved that the sizes of these sets of variables has a bound that decreases by a constant factor with each iteration, so the total time required to simulate the $O(\log n)$ rounds will be linear. ■

To obtain an infinite family of good codes, we will use a family of good expander graphs and good codes known to exist by Theorem 1.4.1.

Theorem 2.5.3 For all ϵ such that $1 - 2H(\epsilon) > 0$, where $H(\cdot)$ is the binary entropy function, there exists an easily constructible family of linear codes with rate $1 - 2H(\epsilon)$ and minimum relative distance arbitrarily close to ϵ^2 such that the parallel decoding algorithm will decode up to a $\frac{\epsilon^2}{64}$ fraction of errors using $O(\log n)$ parallel explicit expander code decoding rounds. This decoding algorithm can be simulated in linear time on a sequential machine.

Proof: Any family of good expander graphs will suffice. For example, we could use those known to exist by Theorem 2.4.7. Let $G_{n,d}$ denote a graph of degree d on n vertices from this family, if such a graph exists. Let λ_d be an upper bound on the eigenvalues other than d in the graphs $G_{n,d}$. Let $B_{n,d}$ denote the edge-vertex incidence graph of $G_{n,d}$.

As we let d grow large in Theorem 2.5.2, the $\frac{\lambda_d}{d}$ terms tend to zero. Moreover, Theorem 1.4.1 tells us that, for sufficiently large block-length d, there are linear codes of rate r and minimum relative distance ϵ if $r \leq 1 - H(\epsilon)$. We use such a code as the subcode in the construction of Theorem 2.5.2. ■

2.5.1 A generalization

Noga Alon has pointed out that the "edge to vertex" construction that we use to construct expander codes is a special case of a construction due to Ajtai, Komlós and Szemerédi [AKS87]. They construct unbalanced expander graphs from regular expander graphs by identifying the large side of their graph with all paths of length k in the original graph and the small side of their graph with the vertices of the original graph. A node identified with a path is connected to the nodes identified with the vertices along that path. The construction used in Theorem 2.5.2 is the special case in which $k = 1$.

Alon suggested that the codes produced by applying the technique of Theorem 2.5.3 to this more general class of graphs can be analyzed by applying Lemma 2.4.14.

Theorem 2.5.4 For all integers $k \geq 2$ and all ϵ such that $1 - kH(\epsilon) > 0$, there exists an easily constructible family of linear codes with rate $1 - kH(\epsilon)$ and minimum relative distance arbitrarily close to $\epsilon^{1+1/(k-1)}$ such that a parallel decoding algorithm will decode some $O(\epsilon^{1+1/(k-1)})$ fraction of errors in $O(\log n)$ parallel decoding rounds, each of which takes constant time. Moreover, this algorithm can be implemented to run in linear time on a sequential machine.

Proof: [Sketch] As in Theorem 2.5.2, we will use the graphs known to exist by Theorem 2.4.7. We will form a code by using the k-path-vertex incidence graph of $G_{n,p+1}$ and by using a Gilbert-Varshamov good code as the subcode. Theorem 1.4.1 implies that for sufficiently large block-length $p+1$, there exist linear codes of rate r and minimum relative distance ϵ provided that $r \leq 1 - H(\epsilon)$. Moreover, as $p+1$ grows large, the terms involving λ in Lemma 2.4.14 goes to zero. If this term were zero, then we would know that every set containing an $\epsilon^{\frac{k+1}{k}}$ fraction of the variables has at least an $\epsilon^{\frac{1}{k}}$ fraction of the constraints as neighbors. Because there are nd^k variables of degree k and n constraints of degree kd^k, Theorem 2.2.2 would imply that no word of weight up to $\epsilon^{\frac{k+1}{k}}$ can be a codeword. However, because λ never actually reaches zero, we can only come arbitrarily close to this bound.

To decode these codes, we modify the parallel explicit expander code decoding algorithm so that each constraint sends a "flip" message to its variables only if the setting of its variables is within ϵ/h of a codeword, for some constant h depending on k. An analysis similar to that in the proof of Theorem 2.5.2 shows that if an α fraction of the variables were corrupted at the start of a round, then at most an

$$\frac{\alpha(k+1)}{h-1} + \frac{\alpha h}{\epsilon}\left(\frac{\alpha h}{\epsilon} + \frac{\lambda_d}{d}\left(1 - \frac{\alpha h}{\epsilon}\right)\right)$$

fraction of the variables will be corrupt after the decoding round. We can choose α to be some fixed fraction of $\epsilon^{1+1/k}$ and a value for h so that this term is less than α.

The idea behind the sequential simulation of this algorithm is the same as appeared in the proof of Theorem 2.5.2. ■

2.6 Notes on implementation and experimental results

We imagine using expander codes in coding situations where long block length is required. For small block lengths, special codes are known that will provide better performance. Expander codes should be especially useful for coding on write-once media, such as Compact Discs, where fast decoding is essential, the time required to encode is not critical, and codes of block length roughly $10,000$ are already being used [Ima90].

If one intends to implement expander codes, we suggest using the randomized construction presented in Sections 2.1.1 and 2.3. We obtained the best performance from graphs that had degree 5 at the variables. It might be possible to get good performance with randomly chosen graphs and slightly more complex constraints, but we have not observed better performance from these.

One drawback of using a randomly chosen graph to generate an expander code is that, while the graph will be a good expander with high probability, we know of no polynomial time algorithm that will certify that a graph has the level of expansion that we need for this construction.[3] On the other hand, it is very easy to perform experiments to test the performance of an expander code and thereby weed out those that do not work well on average.

We now mention a few ideas that we hope will be helpful to those interested in implementing these codes.

- When one chooses a random graph as outlined in Section 2.1.1, there is a good chance that the graph produced will have a "double edge". That is, two edges between the same two vertices. We suggest using a simple heuristic to remove one of these edges, such as swapping its endpoint with that of a randomly chosen edge. If one is choosing a relatively small graph, say of 2,000 nodes, then there is a fair chance that there will be two variables that share two neighbors. Again, we suggest discarding such graphs. In general, if one is choosing a relatively small graph, then there is a reasonable chance that it will have some very small set of vertices with low expansion. We were always able to screen these out by experiment.

- The sequential decoding algorithm presented in Section 2.3.1 can be improved by the same means as many "slope-descent" algorithms. In experiments, we found that a good way to escape local minima was to induce some random errors. The sequential decoding algorithm also benefits from a little randomness in the choice of which variable it flips next. We also found that we could decode more errors if we allow the

[3] Computation of the eigenvalues of the graph does not work because Kahale [Kah93a] has proved that the eigenvalues cannot certify expansion greater than $d/2$.

algorithm to make a limited amount of negative progress. This finding is evidenced in figure 2-4.

- The parallel decoding algorithm presented in Section 2.3.3 seems better suited for implementation in hardware than the sequential algorithm. The performance of this parallel algorithm can be improved by changing the threshold at which it flips variables. An easy improvement is to start the threshold high and decrease it only when necessary. The algorithm performs even better if one only flips the variables that have the maximum number of unsatisfied neighbors. Of course, many hardware implementations will not allow this flexibility. We suggest that the implementer experiment to discover the arrangement of thresholds that performs best in their application.

These codes really are very easy to implement. We implemented these codes in C after a few hours work and set about testing their performance. In each test, we chose a random bipartite graph of the correct size and tested the performance of various decoding algorithms against random errors. For our tests, we never performed encoding—it suffices to examine the performance of the decoding algorithms around the $\vec{0}$ word. We present some results of our experiments so that the reader will know what to expect from these codes. However, we suggest that researchers implement and try them out on the error-patterns that they expect to encounter.

2.6.1 Some experiments

In our experiments, we found that expander codes of the type discussed in Section 2.3 performed best when derived from a randomly chosen graph with degree 5 at the variables. We varied the rates of our codes by changing the degrees of the graphs at the constraints. In Figure 2-3, we present the results of experiments performed on some expander codes of length $40,000$. We began by choosing an appropriate graph at random as described in Section 2.1.1. We then implemented a variation of the sequential decoding algorithm from Section 2.3.1 in which we allowed the algorithm to flip a limited number of variables even if they appeared in only two unsatisfied constraints (we call these “negative progress flips”). We then tested each code on many error patterns of various sizes. For each size, we choose

50, 000 error patterns of this size uniformly at random. Points in the graph indicate the number of errors that a code corrected in all 50, 000 tests. One can see that the expander codes corrected an amazingly large number of errors.

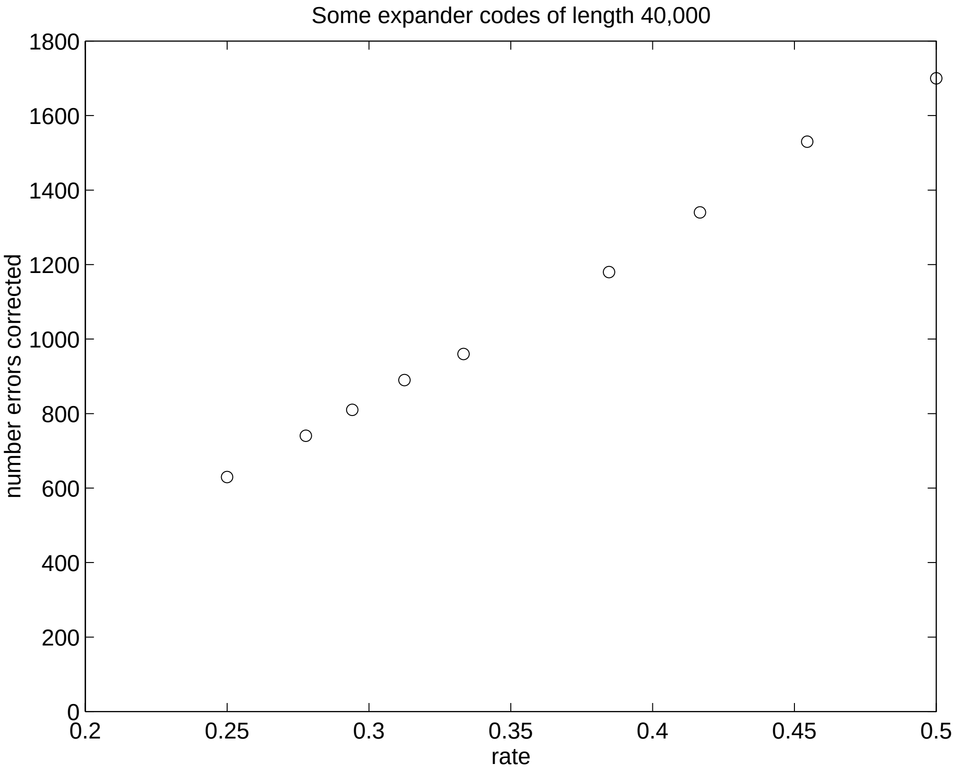

Figure 2-3: Number of errors expander codes could almost always correct. For example, the point in the upper left hand corner of the figure indicates that the code of rate 1/2 corrected all of the 50, 000 patterns of 1, 720 errors on which it was tested.

In Figure 2-4, we compare the performance of a version of the sequential decoding algorithm in which we allow negative progress flips with a version in which we do not. We chose a rate 1/2 expander code of length 40, 000. For each number of errors, we performed 2, 000 tests of each algorithm and counted the number of times that each successfully corrected the errors. While allowing negative progress flips did not seem to have much impact on the number of errors that the algorithms could "almost always" correct, it did greatly increase the chance that the algorithm could correct a given number of errors.

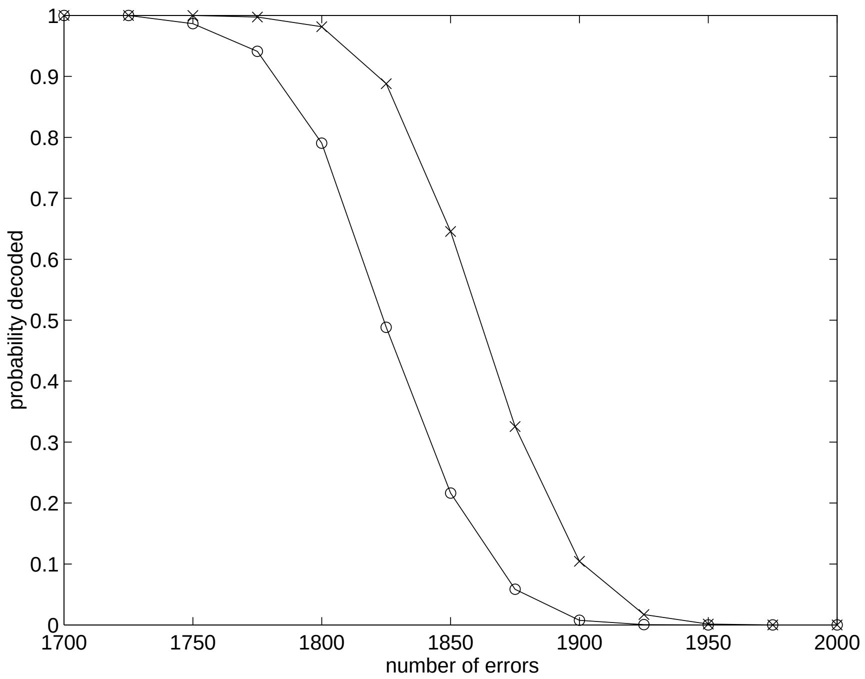

Figure 2-4: A rate 1/2 expander code of length 40,000. Comparison of the probability that the sequential decoding algorithm corrects a given number of randomly chosen errors when we do and do not allow negative progress flips (a negative progress flip occurs when a variable in more satisfied than unsatisfied constraints is flipped).

CHAPTER 3

Linear-time encodable and decodable error-correcting codes

In this chapter, we build on the techniques developed in Chapter 2 to construct asymptotically good families of error-correcting codes that have both linear-time encoding and decoding algorithms. We call these codes *superconcentrator codes* because their encoding circuits resemble Pippenger's construction of superconcentrators [Pip77]. In Section 3.1, we explain why this resemblance is necessary.

To aid in our construction of superconcentrator codes, we introduce the notion of an *error-reducing code*. An error-reducing code has the property that if a decoder receives a partially corrupted codeword, then it can correctly compute a large fraction of its message bits. The fraction of the message bits that the decoder cannot compute should be smaller than the fraction of the bits of the codeword that were corrupted in transmission. In Section 3.2, we construct error-reducing codes that have very fast linear-time encoding and error-reducing algorithms. One could imagine using such a code if one is transmitting data that is already encoded by an error-correcting code. In such a situation, it is not necessary to remove all the errors created in transmission, but it might be advantageous to quickly correct some of them.

In Section 3.3, we demonstrate how to recursively combine error-reduction codes to build our error-correcting superconcentrator codes. The constructions in Sections 3.2 and 3.3 are analogous to those in Section 2.3. The only way that we know of obtaining expander graphs of the quality needed for this construction is to chose the graphs at random.

To build explicit constructions of superconcentrator codes in Section 3.4, we use a construction of error-reduction codes analogous to the construction of expander codes in Section 2.5. The constructions and proofs in Section 3.4 are analogous to those in Section 3.3.

3.1 Motivating the construction

It is not an accident that our linear-size encoding circuits look like superconcentrators. They have to. In this section, we will explain why. While this fact was the motivation behind our construction, one should be able to understand our construction without reading this section.

Consider a circuit C that takes as input some bits $x_1, \ldots, x_m$, and produces as output bits $p_1, \ldots, p_n$ such that the words $x_1, \ldots, x_m, p_1, \ldots, p_n$ form a good error-correcting code. This means that there is a constant δ such that even if we erase[1] any δm of the input bits and any δn of the output bits, we can still recover the erased δm input bits. We will show that this means that there must be gate-disjoint paths from the erased inputs to some subset of the un-erased outputs.

Assume that we cannot find δm vertex-disjoint paths from the erased inputs to the un-erased outputs. Then, Menger's Theorem implies that there is some set of $\delta m - 1$ gates in the circuit such that all paths in the circuit from the erased inputs to the un-erased outputs must go through these gates. This contradicts our assumption that it is possible to recover the values of the erased inputs because there are δm bits of information in the erased input gates, but only $\delta m - 1$ bits of information can get through to the un-erased output gates.

Thus, we see that vertex disjoint paths can be drawn in the underlying graph from any δm input gates into any $(1 - \delta)n$ output gates. While this property is not quite as strong as the property required of superconcentrators, it is sufficiently close that we

[1] Usually, we discuss errors in which a bit's value is flipped. It is also possible to consider the case in which no value is received for some bit. In this case, the decoder knows that the value of the bit has been lost. These errors are called *erasure errors*. Any error-correcting code that can tolerate the first type of error can also tolerate erasure errors. If one is uncomfortable with erasure errors, then just flip a coin and assign its value to each erased bit. One will probably be left with half as many errors of the first type.

decided that the easiest way to create linear-size encoding circuits would be to base them on Pippenger's [Pip77] construction of linear-size superconcentrators.

3.2 Error-reducing codes

In this section, we introduce the concept of an error-reducing code. While we are not sure if this idea will have practical applications, it will be useful for understanding our main construction.

We will define an *error-reducing code* of rate $r < 1$, error-reduction $\epsilon < 1$, and reducible distance $\delta < 1$ to be the image of a function

$$f : \{0,1\}^{rn} \rightarrow \{0,1\}^{n}$$

such that there exists a function

$$g : \{0,1\}^{n} \rightarrow \{0,1\}^{rn}$$

such that for all $x \in \{0,1\}^{rn}$ and $z \in \{0,1\}^{n}$

$$d(f(x), z) \leq \delta n \qquad \Rightarrow \qquad d(x, g(z)) < \epsilon d(f(x), z).$$

f is the encoding function, and g is the decoding (error-reduction) function.

To construct an error-reducing code, we will modify the construction of expander codes from Chapter 2. Let B be an unbalanced bipartite expander graph with n nodes of degree d on one side and $n/2$ nodes of degree $2d$ on the other. We will turn this graph into a circuit by directing all the edges from the large side of the graph to the small side, letting the nodes on the large side be the input gates, and letting the nodes on the small side be parity gates[2] (see Figure 3-1). These parity gates are the outputs of the circuit.

Warning: The pictures that appear in this chapter are very similar to those that appeared in Chapter 2, but their meaning is different! In Chapter 2, the nodes on the right-hand side of

[2] A parity gate computes the sum modulo 2 of its inputs.

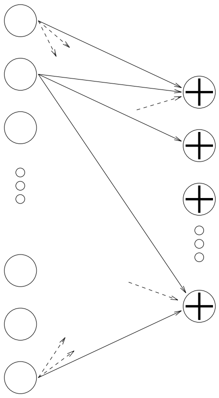

Figure 3-1: *A circuit that encodes an error-reducing code.*

the bipartite graphs represented restrictions of the nodes on the left-hand side. In this chapter, the nodes on the right-hand size of the bipartite graphs have values that are determined by the nodes on the left-hand side.

We now use this circuit to define a code $\mathcal{R}_{n,d}$ of rate 2/3 by placing the message bits at the inputs to the circuit and using the outputs of the parity gates as the check bits. The code we thereby obtain is a horrible error-correcting code: it has a word of weight $d+1$ (see Figure 3-2). However, if the graph is a good expander graph, then this code is

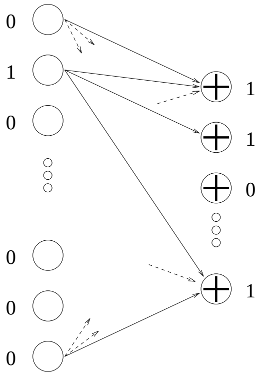

Figure 3-2: *A low-weight codeword: only one input bit is 1, and only those parity gates that read this bit are 1; all others are 0.*

a good error-reducing code. It is obvious that we can encode this code in linear time (we need merely compute the values of the parity gates, each of which has a constant number of inputs). We will now show that it is possible to perform error-reduction on this code in linear time.

The decoder naturally associates each bit of the word that it receives with one of the gates in the encoding circuit. We will say that a parity gate in the circuit is *satisfied* by a word if the bit associated with the gate is the parity of the bits associated with its inputs. Otherwise, we will say that it is *unsatisfied* (see Figure 3-3). The decoder will successively flip the bits associated with the input gates in an effort to decrease the number of unsatisfied parity gates.

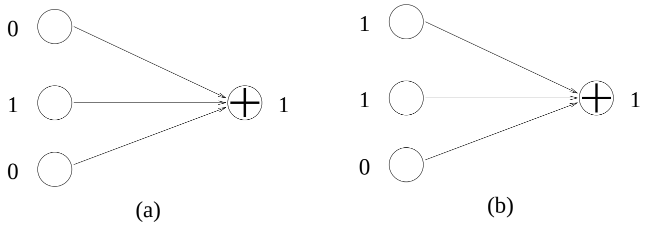

Figure 3-3: *(a) is a satisfied parity gate. (b) is unsatisfied.*

Sequential error-reduction algorithm:

- If there is an input that is an input to more unsatisfied than satisfied parity gates, then flip the value of that input.
- Repeat until no such inputs remain.

This algorithm is essentially the same as the sequential expander-code decoding algorithm of Section 2.3, so it is easy to implement it so that it takes constant time for each iteration. At each iteration, it decreases the total number of unsatisfied parity gates, so it can run for at most a linear number of iterations.

Lemma 3.2.1 Let $\mathcal{R}_{n,d}$ be derived from a degree $(d, 2d)$ bipartite graph between a set of n inputs and $n/2$ parity gates such that all sets of at most αn inputs expand by a factor of at least $(\frac{3}{4}d + 1)$. Assume that the sequential error-reduction algorithm is given a word that resembles a codeword of $\mathcal{R}_{n,d}$ except that at most $\alpha n/2$ of the inputs have been corrupted and at most $\alpha n/2$ of the gates have been corrupted. Then, after the termination of the sequential error-reduction algorithm, at most $\alpha n/4$ of the inputs will be corrupted.

Proof: We will let V denote the corrupted inputs, v the size of V, u the number of unsatisfied parity gates with inputs in V, and s the number of satisfied parity gates with inputs in V. We will view the pair (u, v) as the state of the algorithm. We will first show that if $\alpha n/4 \leq v \leq \alpha n$, then there is some input in more unsatisfied than satisfied parity gates. The expansion of the graph implies that

$$u + s \quad \geq \quad \left(\frac{3}{4}d + 1\right) v. \tag{3.1}$$

Each gate with an input in V accounts for at least one wire leaving V. It is possible that as many as $\alpha n/2$ of the satisfied parity gates with inputs in V are satisfied because they have only one wire from V, but they have been corrupted. The rest must have two wires from V. By counting the dv wires leaving V, we obtain

$$dv \geq u + s + (s - \alpha n/2) \;\Rightarrow\; dv + \alpha n/2 \geq u + 2s \tag{3.2}$$

Combining equations (3.1) and (3.2), we find

$$\begin{aligned} s \quad &\leq \quad \left(\frac{1}{4}d - 1\right) v + \alpha n/2, \qquad \text{and} \\ u \quad &\geq \quad \left(\frac{1}{2}d + 2\right) v - \alpha n/2. \end{aligned} \tag{3.3}$$

When $\alpha n \geq v \geq \alpha n/4$, we have $u > dv/2$, so there must be some input in more unsatisfied than satisfied parity gates.

To show that the algorithm must terminate with $v \leq \alpha n/4$, we show that v must always be less than αn. We assume that when the algorithm begins $v \leq \alpha n/2$ and therefore $u \leq d\alpha n/2 + \alpha n/2$. As the algorithm proceeds, u must steadily decrease. However, if the algorithm is ever in a state (u, v) in which $v = \alpha n$, then equation (3.3) would imply that $u \geq d\alpha n/2 + 3\alpha n/2$, which would be a contradiction.

Thus, the algorithm must always maintain the condition that $v < \alpha n$. This implies that the algorithm cannot terminate unless it is in a state in which $v < \alpha n/4$. ■

We have proved that $\mathcal{R}_{n,d}$ is an error-reducing code of rate 2/3 and error-reduction 1/2.

It is also possible to perform error-reduction on this code in parallel.

Parallel error-reduction algorithm

- For each input, count the number of unsatisfied parity gates to which it is an input.
- For each input that is an input to more unsatisfied than satisfied parity gates, flip the value of that input.

It is easy to implement this algorithm as a constant-depth circuit of linear size.

Lemma 3.2.2 Let $\mathcal{R}_{n,d}$ be derived from a degree $(d, 2d)$ bipartite graph between a set of n inputs and $n/2$ parity gates such that all sets of at most αn inputs expand by a factor of at least $\left(\frac{3}{4} + \epsilon\right) d$, for any $\epsilon > 2/d$. Assume that the parallel error-reduction algorithm is given a word that resembles a codeword of $\mathcal{R}_{n,d}$ except that at most $v \leq \alpha n/2$ of the inputs have been corrupted, and at most $b \leq \alpha n/2$ of the gates have been corrupted. Then, after the execution of the parallel error-reduction algorithm, at most

$$\frac{v + \frac{b}{d}(6 + 8\epsilon)}{1 + 4\epsilon}. \tag{3.4}$$

of the inputs will be corrupted.

Proof: We will let V denote the set of corrupted inputs, F the set of corrupted inputs that fail to flip, and C the set of inputs that were originally clean, but which become corrupted by the parallel error-reduction algorithm. We will let $N(V)$, the neighbors of V, denote the set of parity gates that contain inputs in V. We will let $v = |V|$, $\phi v = |F|$, $\gamma v = |C|$, and $\delta d v = |N(V)|$.

We begin by obtaining a bound on ϕ in terms of δ. Every input in F is an input to at least as many satisfied as unsatisfied parity gates. At most b of these satisfied parity gates are satisfied because they have been corrupted. Thus, at least $\frac{d\phi v}{2} - b$ of the wires leaving F end in a parity gate that contains a wire from another element of V. Thus, the set V can have at most

$$dv - \frac{d\phi v}{4} + \frac{b}{2}$$

neighbors. Because we have set the number of neighbors of V to be δdv, we obtain

$$\begin{aligned} \delta dv &\leq dv - \frac{d\phi v}{4} + \frac{b}{2} \\ \frac{dv\phi}{4} &\leq (1-\delta)dv + \frac{b}{2} \\ \phi &\leq 4(1-\delta) + \frac{2b}{dv} \end{aligned} \tag{3.5}$$

Next, we will bound γ in terms of δ by showing that

$$\gamma < \frac{\delta - (\frac{3}{4}+\epsilon) + \frac{b}{dv}}{\frac{1}{4}+\epsilon}. \tag{3.6}$$

Assume by way of contradiction that this is false, and let C' be a subset of C of size exactly $\frac{\delta-(\frac{3}{4}+\epsilon)+\frac{b}{dv}}{\frac{1}{4}+\epsilon}$. Each input in C' is an input to at least $d/2$ unsatisfied parity gates. At most b of these gates can be unsatisfied because their parity gate has been corrupted. The others must be unsatisfied because they have an input in V. Thus, the number of parity gates containing inputs in the set $V \cup C'$ is at most

$$d\delta v + \frac{d}{2}|C'| + b.$$

We will set $\epsilon > 2/d$, which will imply that $|C' \cup V| < \alpha n$, so this set must expand by a factor of at least $(\frac{3}{4}+\epsilon)$. Thus, we obtain

$$\left(\frac{3}{4}+\epsilon\right)|C' \cup V| < d\delta v + \frac{d}{2}|C'| + b,$$

which contradicts our assumption about the size of C'.

By combining equations (3.5) and (3.6), we find that the number of corrupted inputs after an execution of the parallel error-reduction algorithm is at most

$$\begin{aligned} (\phi+\gamma)v &\leq v\frac{\delta - (\frac{3}{4}+\epsilon) + \frac{b}{dv}}{\frac{1}{4}+\epsilon} + 4(1-\delta) + \frac{2b}{d} \\ &= \frac{v\left(\frac{1}{4}+3\epsilon-4\epsilon\delta\right) + \frac{b}{d} + \left(\frac{1}{4}+\epsilon\right)\frac{2b}{d}}{\frac{1}{4}+\epsilon} \\ &< \frac{v + \frac{b}{d}(6+8\epsilon)}{1+4\epsilon}. \end{aligned}$$

■

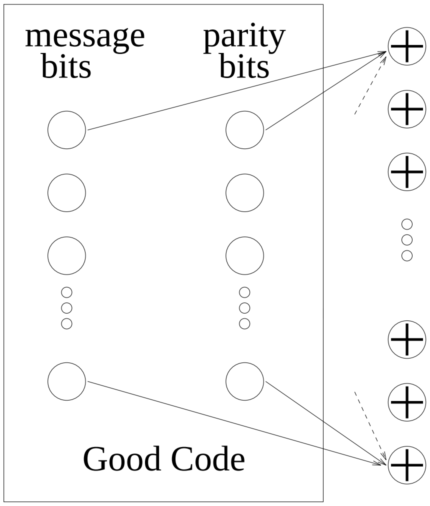

Figure 3-4: A good code embedded in an error-reduction code.

We can iterate this algorithm a constant number of times in order to further reduce the number of corrupted input bits.

3.3 Error-correcting codes

The main idea in our construction of linear-time encodable and decodable error-correcting codes is to use the linear-time encodable and decodable error-reducing codes recursively. Imagine what happens when we take a good error-correcting code of length n and use the words of this code as the message bits of an error-reducing code (See Figure 3-4). We now have an error-correcting code of greater length with a higher error-tolerance; however, it has the same number of message bits as the original code. To increase the number of message bits in the code, we will create a new set of more message bits, and encode these message bits in an error-reduction code. We then use the check bits of this error-reduction code as the message bits of the error-correcting code we just constructed (See Figure 3-5).

When we apply this construction recursively, we find that not only have we defined the error-correcting code, but we have created a linear-size circuit that encodes the error-correcting code. Moreover, the output of each gate of this circuit is a check bit of the error-correcting code. This will be true with one small exception: we should choose a better base case for our recursion. The performance of our code will be dictated by the

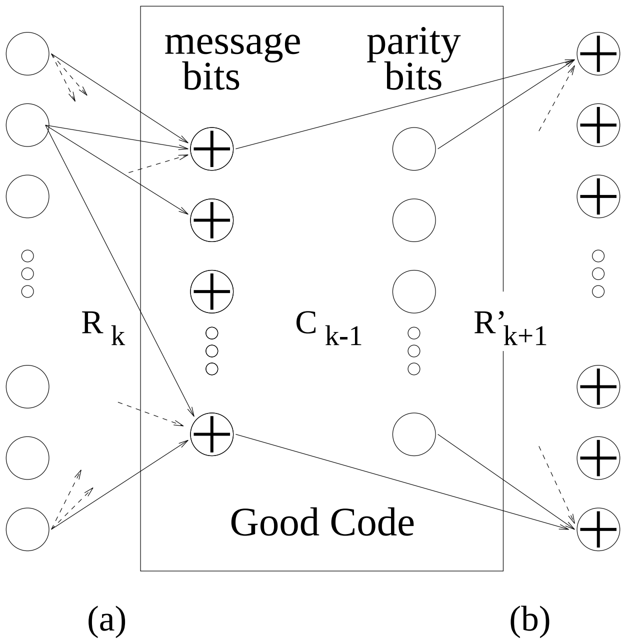

Figure 3-5: *The recursive construction of C_k. (a) is the error-reducing code on the new message bits. (b) is the error-reducing code placed on top of the error-correcting code.*

quality of the code that we choose as our base. Thus, we may want to choose a particularly good code of small block length. Another constraint is that we don't know that we will be able to find good expander graphs of small size. However, there definitely is some constant size after which we will be able to find good expander graphs, and which we will choose to be the block length of our base code.

We will now present a formal description of one family of superconcentrator codes. We provide this description by describing the encoding circuits for these codes.

Description of superconcentrator codes:

- Choose absolute constants b and d.
- Choose a code $\mathcal{C}_b$ of length $4 \cdot 2^b$, rate $1/4$, and as large minimum distance as possible.
- Let C_b be a circuit that takes 2^b bits as input and produces $3 \cdot 2^b$ bits as output so that these bits taken together form a codeword of $\mathcal{C}_b$. (The 2^b inputs bits are the message bits of the code, and the others are the check bits.)
- For $k > b$, let R_k and R'_k be circuits that encode an error-reduction code of rate $2/3$ with

2^k message bits and $2^k/2$ check bits, as described in Section 3.2.

- To form circuit C_k from C_{k-1}, take a copy of R_k and use the inputs of R_k as the inputs of C_k. Add C_{k-1} to the circuit by identifying the output gates of R_k with the inputs of C_{k-1}. Finally, attach a copy of R'_{k+1} by identifying all the input and output gates of the copy of C_{k-1} with the inputs of R'_{k+1}. The output gates of C_k will be all the input and output gates of C_{k-1} along with the output gates of R'_{k+1} (See Figure 3-5).
- Let $\mathcal{C}_k$ be the rate 1/4 code obtained by taking 2^k message bits, feeding them into C_k, and using the $3 \cdot 2^k$ output bits as the parity checks of the code.

We can decode these codes in linear sequential time.

Sequential superconcentrator code decoding algorithm:

- If $k = b$, then decode $\mathcal{C}_b$ using an arbitrary decoding algorithm.
- If $k > b$, then apply the sequential error-reduction algorithm to the nodes in the R'_{k+1}. Now, recursively decode the nodes in the copy of C_{k-1} using the sequential decoding algorithm for C_{k-1}. Finish by applying the sequential error-reduction algorithm to the copy of R_k.

Theorem 3.3.1 If the superconcentrator code $\mathcal{C}_k$ is constructed from degree $(d, 2d)$ graphs such that in each graph, every at most α fraction of inputs expands by a factor of at least $(\frac{3}{4}d + 1)$, and if $\mathcal{C}_b$ is chosen to be a code in which any $\alpha/2$ fraction of errors can be corrected, then the sequential superconcentrator code decoding algorithm will correct up to an $\alpha/8$ fraction of errors and will run in linear time.

Proof: We assume that there are at most $\alpha 2^k/2$ errors in the nodes of C_k. By Lemma 3.2.1, after we apply the sequential error-reduction algorithm R'_{k+1}, there will be at most $\alpha 2^k/4$ errors in the nodes of the copy of C_{k-1}. We can now assume by induction that the decoding algorithm for C_{k-1} will correct all the errors in its input and output nodes. As the input nodes of C_{k-1} are now the check bits of the error-reduction code $\mathcal{R}_k$ corresponding to the message originally contained at the inputs of the copy of R_k, and there are at most $\alpha 2^k/2$ errors in these message bits, we can use the sequential error-reduction algorithm to correct all the errors in the input nodes of R_k (This can be easily observed from the proof of Lemma 3.2.1, or from the analysis of the sequential expander code decoding algorithm in Theorem 2.3.1). Since the inputs of R_k are the inputs of C_k, we have removed all the errors from the message bits of the code.

To see that this algorithm runs in linear time, observe that each error-reduction step runs in time that is linear in the number of bits that it is acting on, and each step acts on half as many bits as the previous step did. ■

It is easy to see that there is a constant α that satisfies the requirements of Theorem 3.3.1, but we will not attempt to optimize the constant here. We will note that the main constraint on the constant is in the analysis of the quality of expansion obtained by a randomly chosen graph.

Remark 3.3.2 We have only constructed codes of lengths 2^k where k is an integer. It is easy to use similar techniques to construct codes of other lengths.

We need to be slightly trickier to decode the superconcentrator codes in parallel logarithmic time. The problem that we must overcome is that if we iterate the parallel error-reduction algorithm enough times to remove all the errors from the input bits of C_i, we will need to go through $O(i)$ iterations. If we did this for each C_i, then we would have an $O(log^2 n)$ time algorithm. To overcome this problem, we will perform the error-reductions from the input bits of C_{i-1} to the input bits of C_i *simultaneously* for all i. Thus, while the input gates of C_{i-1} are being used to reduce the errors in the input gates of C_i, the input gates in C_i are being used to reduce the errors in the input gates of C_{i+1}.

In order to show that the reduction of errors of the bits of C_i using the output bits of R'_{i+1} works, we will assume $\epsilon = \frac{4}{d} + \epsilon'$, for $\epsilon' > 0$ and $d > 16$. We now wish to observe that if $b \leq \alpha n/2$, and $\alpha n/4 \leq v \leq \alpha n/2$, then after one round of the parallel error-reduction algorithm, v will decrease by a constant multiplicative factor. From equation (3.4), we can see that this constant factor will be bounded by the decrease that occurs when $v = \alpha n/4$ and $b = \alpha n/2$. By plugging these values into equation (3.4), we can see that this constant is less than 1. We now know that if $v \leq \alpha n/2$ and $b \leq \alpha n/2$, then after a constant number of rounds, we will have $v \leq \alpha n/4$. Let c be this constant.

To prove the correctness of the error-reductions on the input nodes of the C_i's, we will assume that there is a $w \leq \alpha n/2$ such that $v \leq w$ and $b \leq w/2$. By substituting into equation (3.4), we find that after one decoding round the number of corrupted inputs is bounded by

$$w\left(\frac{1+\left(\frac{3}{d}+\frac{16}{d^2}+\frac{4\epsilon'}{d}\right)}{1+\frac{16}{d}+4\epsilon'}\right) \quad < \quad w\left(1-\frac{\frac{11}{d}}{1+\frac{16}{d}+4\epsilon'}\right).$$

Let $\gamma = \left(1-\frac{\frac{11}{d}}{1+\frac{16}{d}+4\epsilon'}\right)$.

We can now state the parallel superconcentrator code decoding algorithm.

Parallel superconcentrator code decoding algorithm:

- For $i = k-1$ to b: Apply c rounds of the parallel error-reduction algorithm using the inputs and outputs of C_i as the message bits, and the outputs of R'_{k+1} to which they are attached as the check bits.
- Decode the errors in C_b using any decoding algorithm.
- For $\log_{1/\gamma} 2^k$ rounds: Apply the parallel error-reduction algorithm to the copy of R_i between the inputs of C_i and the inputs nodes of C_{i+1}, simultaneously for all $b \leq i \leq k-1$.

Theorem 3.3.3 If the superconcentrator code $\mathcal{C}_k$ is constructed from degree $(d, 2d)$ graphs such that in each graph, every at most α fraction of inputs expands by a factor of at least $(\frac{3}{4}+\frac{4}{d}+\epsilon')d$, for some $\epsilon' > 0$ and $d > 16$, and if $\mathcal{C}_b$ is chosen to be a code in which any $\alpha/2$ fraction of errors can be corrected, then the parallel superconcentrator code decoding algorithm will correct up to an $\alpha/8$ fraction of errors in logarithmic time with a linear number of processors.

Proof: We begin by assuming that there are at most $\alpha 2^k/2$ errors in the bits of C_k. After we apply c rounds of the parallel error-reduction algorithm to R_{k+1}, there will be at most $\alpha 2^k/4$ errors in the bits of the copy of C_{k-1}. Similarly, after we have finished the i-th stage of the algorithm, there will be at most $\alpha 2^k/2^{i+1}$ errors in the bits of C_{k-i}. Thus, C_b will have fewer than $\alpha 2^b/2$ errors, so the decoding algorithm for C_b will correct all the errors in C_b.

We can now move on to the decoding of the input bits of the C_i's. We have already observed that the input bits of C_i have at most $\alpha 2^i/2$ errors and that the input bits of C_b are free of error. Thus, after we apply one round of the error-reduction algorithm simultaneously to all of the R_i's, the input bits of C_i will have at most $\gamma\alpha 2^i/2$ errors. Similarly, after we apply the error-reduction algorithm for $\log_{1/\gamma} 2^k$ rounds, there will be no more errors in any of the input nodes of C_k. ■

3.4 Explicit Constructions

The relation between our first constructions of superconcentrator codes and our explicit constructions is analogous to the relation between our constructions of expander codes in Sections 2.3 and 2.5. We will begin by generalizing our construction of error-reducing codes.

If we view the inputs to one of the parity gates in Section 3.2 as the message bits of a code, then we can view the parity gate as computing a check bit associated with those message bits. This bit is chosen so that when it is considered with the input bits, these bits form a word in the code of even weight words. To generalize this construction, we will associate a collection of parity gates with each node on the small side of the bipartite graph, and we will connect them to their inputs so that these gates compute the check bits of a codeword in some more complex error-correcting code. We call such a collection of parity gates a *cluster*.

As in Section 2.5, we will make use of a family of good expander graphs, such as those described in Section 2.4.

We now define the circuits $R(G, \mathcal{C})$ that will encode our explicit error-reduction codes.

Explicit error-reduction codes:

- Let G be a d-regular graph on n vertices, and let $\mathcal{C}$ be a linear error-correcting code of block-length l with d message bits. Let B be the edge-vertex incidence graph of G.
- The circuit $R(G, \mathcal{C})$ will have $dn/2$ input gates and $(l-d)n$ parity gates.
- Each node on the large side of B will correspond to an input of $R(G, \mathcal{C})$. The parity gates are arranged into clusters of size $l-d$, and each cluster is identified with one of

the nodes on the small side of B. The input gates that are neighbors of a cluster will be called the inputs of the cluster.

- The parity gates are connected to the input gates so that for each cluster, if the inputs of that cluster are the message bits of a codeword of $\mathcal{C}$, then the parity gates in the cluster compute the check bits of that codeword.

- Let $\mathcal{R}(G,\mathcal{C})$ denote the code obtained by using the inputs of $R(G,\mathcal{C})$ as message bits and the outputs of the circuit as check bits.

To prove that $\mathcal{R}(G,\mathcal{C})$ is a good error-reduction code if G is a good expander graph, we will use Lemma 2.4.6.

To perform error-reduction on $\mathcal{R}(G,\mathcal{C})$, we will associate each bit of a received word with an input or gate of $R(G,\mathcal{C})$, as we did in Section 3.2.

Parallel explicit error-reduction algorithm:

- In parallel, for each cluster, if the bits associated with the inputs and gates of a cluster are within $\frac{\epsilon}{6}$ of a codeword of $\mathcal{C}$, then send a “flip” message to every input that needs to be flipped to obtain that codeword.

- In parallel, every input that receives the message “flip”, flips its value.

We can now prove a lemma for our explicit construction that is analogous to Lemma 3.2.2.

Lemma 3.4.1 Let $\{G_{n,d}\}$ be a family of good expander graphs. There exist constants d and g such that if the parallel explicit error-reduction algorithm is given as input a word that resembles a codeword of $\mathcal{R}(G_{n,d},\mathcal{C})$ except that for some $w \leq \frac{\epsilon^2}{g} n \frac{d}{2}$

- at most $\frac{w}{2} \leq v \leq 2w$ inputs are corrupted and at most $b \leq w$ parity gates are corrupted, then after the execution of the algorithm, the number of corrupted inputs will decrease by a constant multiplicative factor, and

- if at most $v \leq \frac{w}{2}$ of the inputs are corrupted and at most $b \leq w$ of the parity gates are corrupted, then after the execution of the algorithm, at most $\frac{w}{2}$ inputs will be corrupted.

Proof: Let V be the set of v corrupted inputs. Set α and β so that $v = \alpha n\frac{d}{2}$ and $b = \beta n\frac{d}{2}$.

We will say that a cluster is *confused* if it sends a "flip" message to an input that is not corrupt. In order for a cluster to be confused, it must have at least $\frac{5\epsilon}{6}d$ corrupt inputs and gates. Thus, there can be at most

$$\frac{n\frac{d}{2}(2\alpha+\beta)}{\frac{5}{6}\epsilon d}$$

confused clusters. Each of these can send at most $\frac{1}{6}\epsilon d$ "flip" messages, so at most

$$\frac{n\frac{d}{2}(2\alpha+\beta)}{\frac{5}{6}\epsilon d}\cdot\frac{1}{6}\epsilon d = \frac{n\frac{d}{2}(2\alpha+\beta)}{5}$$

uncorrupted inputs can receive "flip" signals.

We will call a cluster *unhelpful* if it has a node of V among its inputs, but it fails to send a "flip" signal to that node. There can be at most

$$\frac{n\frac{d}{2}(2\alpha+\beta)}{\frac{1}{6}\epsilon d}$$

unhelpful clusters. By Lemma 2.4.6, there can be at most

$$n\frac{d}{2}\left(\left(\frac{6\alpha+3\beta}{\epsilon}\right)^2+\left(\frac{6\alpha+3\beta}{\epsilon}\right)\frac{\lambda_d}{d}\right)$$

inputs both of whose neighbors are unhelpful clusters.

The total number of corrupted inputs after the algorithm is run will be at most

$$n\frac{d}{2}\left(\frac{2\alpha+\beta}{5}+\left(\frac{6\alpha+3\beta}{\epsilon}\right)^2+\left(\frac{6\alpha+3\beta}{\epsilon}\right)\frac{\lambda_d}{d}\right).$$

Set $w = \frac{\epsilon^2}{g}n\frac{d}{2}$. We want to show that

$$\frac{2\alpha+\beta}{5}+\left(\frac{6\alpha+3\beta}{\epsilon}\right)^2+\left(\frac{6\alpha+3\beta}{\epsilon}\right)\frac{\lambda_d}{d} < \alpha,$$

for $\beta \in [0, \frac{\epsilon^2}{g}]$ and $\alpha \in [\frac{\epsilon^2}{2g}, \frac{2\epsilon^2}{g}]$. Because the function in consideration is decreasing in β, and quadratic in α with positive coefficients, it suffices to consider the function at the points $\alpha \in \left\{\frac{\epsilon^2}{2g}, \frac{2\epsilon^2}{g}\right\}$ and $\beta = \frac{\epsilon^2}{g}$. For $\alpha = \frac{\epsilon^2}{2g}$, we need

$$\frac{\epsilon^2}{10g} > \left(\frac{6\epsilon}{g}\right)^2 + \frac{6\epsilon}{g} \cdot \frac{\lambda_d}{d},$$

and for $\alpha = \frac{2\epsilon^2}{g}$, we need

$$\frac{\epsilon^2}{g} > \left(\frac{15\epsilon}{g}\right)^2 + \frac{15\epsilon}{g} \cdot \frac{\lambda_d}{d}.$$

We now see that, given ϵ, we can choose g and then d so that both of these inequalities hold. For lower values of w, the claim follows from a similar analysis.

To prove the second claim, it suffices to observe that the function in question is decreasing in α for $\alpha > 0$. ∎

We will find it convenient to let $\mathcal{C}$ be a code of length $5d/4$ and rate $4/5$. By Theorem 1.4.1, there exists a code with length $\frac{5}{4}d$, rate $\frac{4}{5}$, and relative minimum distance ϵ for $H(\epsilon) \leq 1/5$.

To obtain an explicit construction of superconcentrator codes, we will need expander graphs of very particular sizes. Fortunately, we demonstrated in Section 2.4.1 how to construct good expanders of every sufficiently large number of vertices. By Proposition 2.4.19 and Remark 2.4.20, we can construct bipartite graphs between any number of degree-2 vertices and a set of vertices in which each vertex has degree $d-1$ or d and such that the graphs have expansion properties like those graphs used in Lemma 3.4.1. The effect of the differing degrees of the vertices on the small side of the graph can be made negligible by slightly altering some of the codes associated with some of these vertices. Thus, we can assume that we can obtain good error-reduction codes of any sufficiently large size, and that they perform as described in Lemma 3.4.1.

Explicit superconcentrator codes:

- Choose a family of good expander graphs, $\mathcal{G} = \{G_{n,d}\}$ such that the subfamily for each d contains a dense subsequence.
- Choose a constant d and a code C_d of rate $4/5$, length $5d/4$ and minimum relative distance

ϵ where $\epsilon \geq H(1/5)$. (As described in the previous paragraph, we will assume that $\mathcal{G}$ contains a graph of every sufficiently large number of vertices in which every vertex has degree d or $d-1$.)

- Choose a constant b and a code $\mathcal{C}_b$ of length $4 \cdot 2^b$, rate $1/4$, and as large minimum distance as possible. Let C_b be a circuit that takes 2^b bits as input and produces $3 \cdot 2^b$ bits as output so that these bits taken together form a codeword of $\mathcal{C}_b$. (The 2^b inputs bits are the message bits of the code, and the others are the check bits.)
- For $k > b$, let R_k and R'_k be the circuits $R(G_{2^k,d}, \mathcal{C}_d)$ defined earlier in this section. We have chosen the code $\mathcal{C}_d$ so that this circuit has 2^k input gates and $2^k/2$ parity gates (If the graph isn't d-regular, then we might have to modify the codes at the clusters slightly to make sure that we have $2^k/2$ parity gates, but this effect is insignificant).
- To form circuit C_k from C_{k-1}, take a copy of R_k and use the inputs of R_k as the inputs of C_k. Add C_{k-1} to the circuit by identifying the output gates of R_k with the inputs of C_{k-1}. Finally, attach a copy of R'_{k+1} by identifying all the input and output gates of the copy of C_{k-1} with the inputs of R'_{k+1}. The output gates of C_k will be all the input and output gates of C_{k-1} along with the output gates of R'_{k+1}.
- Let $\mathcal{C}_k$ be the code obtained by taking 2^k message bits, feeding them into C_k, and using the $3 \cdot 2^k$ output bits as the parity bits of the code.

Lemma 3.4.1 implies that there is a constant c such that if $v \leq w$ and $b \leq w$, where $w \leq 2^k \epsilon^2/g$, then after c executions of the parallel explicit error-reduction algorithm, we will have at most $w/2$ corrupted inputs. Similarly, there exits a constant $\gamma < 1$ such that if $v \leq 2w$ and $b \leq w$, then after the execution of the error-reduction algorithm there will be at most γv corrupted inputs.

Parallel explicit error-correction algorithm:

- Identical to the algorithm presented in Section 3.3, but using the definitions of C_k and R_k presented in this section.

Theorem 3.4.2 There exist settings of the parameters of the constructions of the rate $1/4$ superconcentrator codes $\mathcal{C}_k$ presented in this section, as well as settings of ϵ and g, such that the parallel explicit superconcentrator code decoding algorithm will correct up to an $\epsilon^2/4g$ fraction of errors in logarithmic time with a linear number of processors. Moreover, this algorithm can be simulated in linear sequential time.

Proof: We use Theorem 2.4.7, Theorem 2.4.15, Proposition 2.4.19, and Remark 2.4.20 to show that there exist expander graphs in the appropriate sizes to construct the 2/3-rate error-reduction codes R_k. The similarity of Proposition 2.4.19 to Lemma 2.4.6 implies that these error-reduction codes will perform as described in Lemma 3.4.1. The rest of the proof of the first claim is identical to the proof of Theorem 3.3.3.

By keeping track of which clusters of parity gates are "unsatisfied", it is fairly simple to simulate this algorithm in linear time on a sequential machine (assuming that pointer references have unit cost). The idea is similar to that used in Theorem 2.5.2. ■

Remark 3.4.3 One can vary this construction to obtain asymptotically good linear-time encodable and decodable codes of any constant rate. To achieve rates exceeding $1/4$, one need merely vary the ratios of the number of gates between successive levels of the construction and possibly omit the last few error-reduction codes.

Remark 3.4.4 We can replace the assumption that pointer references have unit cost with the assumption that retrieving $O(\log n)$ bits stored at a pointer has $O(\log n)$ cost. We do this by concatenating the superconcentrator codes with some good code of length $O(\log n)$. The inner code can be decoded by $O(n/\log n)$ table look-ups (or it can be decoded recursively) and the outer code now contains $O(n/\log n)$ symbols of length $O(\log n)$. The outer code can now be decoded in linear time using $O(n/\log n)$ pointer references if we use a natural generalization of the superconcentrator codes to alphabets of size $O(\log n)$.

3.5 Some thoughts on implementation and future work

Superconcentrator and expander codes will probably only be useful in coding schemes using long block lengths; for small block lengths, many better special codes are known. The tradeoff of rate versus error tolerance of the superconcentrator codes that we have constructed is not nearly as good as the tradeoff obtained by the expander codes constructed in Chapter 2.

In our view, we have added a lot of extra redundant information in order to make the codes encodable and decodable in linear time. Thus, we do not feel that it would be appropriate to use these codes on a channel in which bandwidth is at a premium. However, if one has a fast channel with bandwidth to spare, then it might be reasonable to use the superconcentrator codes that we have constructed here. In particular, if one has a fast channel, then the computational work needed to encode and decode the error-correcting codes used can be a greater bottleneck in communication than the redundancy of the code. For an analysis of potential applications of long block length error-correcting codes, we direct the reader to [Bie92].

Those who desire to implement these codes should consider the issues discussed in Section 2.6. They should also be aware that our construction was optimized for ease of explication and that there are many parameters that one can vary which we have not discussed. For example, it might be advantageous to use error-reduction codes of rates other than 2/3 in the recursive construction.

However, the reason that we did not attempt to optimize the tradeoff of rate versus error tolerance in our construction is that we believe that a new idea will be required to construct codes of rate and error tolerance competitive with the Gilbert-Varshamov bound. That is our goal: to obtain linear-time encodable and decodable error-correcting codes with rate and error tolerance as good as the best known codes, whatever their algorithmic efficiency. Our feeling is that this goal is achievable and that the work in this chapter constitutes an important step towards it, but that it cannot be achieved by simple modifications of our current construction.

CHAPTER 4

Holographic Proofs

A holographic proof system is a system of writing and checking proofs in which one can probabilistically check the validity of a proof by examining only a few of its bits. If someone tells you that they have written a holographic proof of a theorem, then there is a simple procedure by which you can randomly choose a few bits of the proof to examine, perform a simple computation on these bits, and decide whether to accept or reject the proof. You will always accept a valid holographic proof. Conversely, if the probability that you accept is greater than 1/2, then the proof that you examined must be very close to a holographic proof of the theorem; thus, a proof of the theorem does exist. You can repeat this test a few times to obtain an arbitrarily high degree of confidence that the theorem is true. Remarkably, any theorem that has a conventional proof has a holographic proof.

Constructions of holographic proofs[1], which are also known as *transparent proofs* and *probabilistically checkable proofs*, were developed in a series of papers [BFL91, BFLS91, FGL$^+$91, AS92b, ALM$^+$92] which culminated in the following theorem:

Theorem 4.0.1 [PCP-Theorem] Let E be an asymptotically good error-correcting code of

[1] Some authors use the terms *holographic* and *transparent* to describe proof systems in which the theorem to be proved is presented in an encoded format. This encoding is necessary if the proof checker is to run in polylogarithmic time or read only a constant number of bits of the theorem. They use the term *probabilistically checkable* to describe systems in which the proof checker is allowed to read the entire statement of the theorem and run in polynomial time. The purpose of the encoded theorems is discussed further in Section 4.4.

minimum relative distance δ such that there is a polynomial-time algorithm that will determine whether a given word is a codeword of E. Then, for any constant k, there exists a probabilistic algorithm V that expects $k+2$ inputs, $(Y_0, Y_1, \ldots, Y_k, \Pi)$, and runs in time $\log^{O(1)} n$ such that

- V only reads a constant number of bits from each of its $k+2$ inputs;
- if $Y_0 = E(C)$ for some circuit C that expects k inputs of lengths $n_1, \ldots, n_k$ and if $Y_i = E(X_i)$ where $|X_i| = n_i$, for each i, and C accepts on input $(X_1, \ldots, X_k)$, then there is a string Π such that V accepts on these inputs with probability 1; moreover, this string Π can be computed from C and $X_1, \ldots, X_k$ in time $|\Pi| \log^{O(1)} |\Pi|$;
- if the probability that V accepts on inputs $(Y_0, \ldots, Y_k, \Pi)$ is greater than $1/2$, then there exists a circuit C that accepts k inputs of lengths $(n_1, \ldots, n_k)$ and strings $X_1, \ldots, X_k$ of the corresponding lengths such that C accepts on input $(X_1, \ldots, X_k)$ and $d(E(C), Y_0) < \delta/3$ and $d(E(X_i), Y_i) < \delta/3$ for each i;
- V only uses $O(\log |C|)$ random bits; and, Π has size $|C|^{O(1)}$.

We will explore why this theorem takes the form that it does later in this chapter. To see how it implies a statement similar to the informal statement given at the start of this section, we draw the following corollary:

Corollary 4.0.2 There exists a probabilistic polynomial-time turing machine V that accepts two inputs, a description of a circuit C and a witness Π. V reads only a constant number of bits of its input Π and uses only $O(\log |C|)$ random bits. If there is an assignment that satisfies C, then there is an input Π that will cause V to accept with probability 1. Conversely, if there is an assignment of Π that causes V to accept with probability greater than $1/2$, then C has a satisfying assignment.

The witness Π in this corollary is a holographic proof that C has a satisfying assignment. To derive our assertion about proofs of theorems, we observe that the problem of deciding whether any statement has a proof of a given length can be converted into a problem of deciding whether a certain circuit has a satisfying assignment. Using the techniques of [BFLS91], one can show that the size of this circuit will be greater than the size of the statement of the theorem and its proof by at most a poly-logarithmic factor.

The main contribution of this chapter, Theorem 4.6.1, is a strengthening of Theorem 4.0.1 in which the holographic proof, Π, has size $O(|C|^{1+\epsilon})$, for any $\epsilon > 0$. The size of the proof will depend on the number of queries that the proof checker makes. If we allow the proof checker to make as many as $O(\log |C|)$ queries, then we will be able to construct proofs of size $|C|(\log |C|)^{O(\log \log |C|)}$.

4.0.1 Outline of Chapter

Our construction of holographic proofs follows the general plan used in [BFLS91]. The authors of that paper explained how to construct holographic proofs of size $O(|C|^{1+\epsilon})$, for any $\epsilon > 0$, in which the proof checker reads $O(\log |C|)$ bits of the proof. In order to construct such small proofs, they first had to develop an efficient way to translate any proof into a simple, well-structured format. In Section 4.3.1, we present a simpler but slightly weaker means of achieving this translation. The framework presented in this section should be sufficient for most applications. In Section 4.6, we explain how our techniques can be fully integrated with the framework of [BFLS91].

Our holographic proofs are built from special polynomial error-correcting codes that have efficient checkers and verifiers. In Section 4.1, we define checkers and verifiers, and develop the terminology we will use to discuss them.

Section 4.2 is devoted to defining the polynomial codes that we use and to demonstrating that they have the checkers and verifiers that we desire. The main statement of this section, Theorem 4.2.19, is an improved analysis of the "low-degree testing" used in Arora and Safra [AS92b]—it shows that the testing works over a domain whose size is linear in the degree being tested. We present the material in this section using the language we developed in Section 4.1.

In Section 4.3, we combine the analysis from Section 4.2 with ideas from [BFLS91], [AS92b], and [Sud92] to develop relatively simple holographic proofs that can be checked by examining a constant number of segments of the proofs, each of size $\sqrt{n} \log^{O(1)} n$. The proofs in this system can be used to provide proofs of many types of statments, including circuit satisfiability. This section has three parts. In the first, we present the NP-complete problem upon which we base our proof system. In Sections 4.3.2 and 4.3.3, we provide an algebraic description of the problem. The key to this description is a novel algebraic description of a de Bruijn graph, Lemma 4.3.5. This algebraic description is combined with the techniques of Section 4.2 to construct holographic proofs in Section 4.3.4.

In Section 4.4, we recursively apply the holographic proof system of Section 4.3 to itself to obtain the first variation of our construction of nearly linear size holographic proofs, Theorem 4.4.1. The recursion that we use differs from those in [AS92b] and [ALM$^+$92] in that it does not need consistency checking. We do not show that the proofs in this first variation have checkers that run in poly-logarithmic time. In Theorem 4.5.1, we show that there is an easily computed table of information such that, if the proof checker has access to this table, then it can run in poly-logarithmic time. Essentially, this table captures computations that the proof checker would perform regardless of its input. That is the second variation of our construction. In Section 4.6, we use a technique from [BFLS91] to construct checkers that do not need such a table. This statement, Theorem 4.6.1, is the final variation of our main theorem. We conclude by stating a weaker but simpler corollary of this theorem.

Since the history of the development of holographic proofs is rather involved, we do not explain it here, but refer the reader to one of

- Sanjeev Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems.* PhD thesis, U.C. Berkeley, Aug. 1994.

- László Babai. Transparent proofs and limits to approximation. In *First European Congress of Mathematics: (Paris, July 6–10, 1992)*, volume 2, pages 31–92. Birkhäuser, 1994.

- Oded Goldreich. A taxonomy of proof systems. *SIGACT News*, 24(4)—25(1), December 1993—March 1994.

- Madhu Sudan. *Efficient checking of polynomials and proofs and the hardness of approximation problems.* PhD thesis, U.C. Berkeley, Oct. 1992.

4.1 Checkable and verifiable codes

In this section, we define checkable and verifiable codes. Such codes have played an important role in the construction of holographic proofs and were used implicitly in the work of [LFKN90] and [BFL91]. However, we first saw them explicitly defined in [Bab94] and [BF93].

Definition 4.1.1 A *checker* for a family of codes $\{\mathcal{C}_i\}$ of lengths $\{n_i\}$ and relative minimum distance δ over an alphabet Σ is a probabilistic algorithm such that

- The checker accepts each word of $\mathcal{C}_i$ with probability 1.
- If the probability that the checker accepts the word w of length n_i is greater than $1/2$, then there is a unique codeword of $\mathcal{C}_i$ of relative minimum distance at most $\delta/3$ from w.

We will usually measure the performance of a checker by the number of bits of its input that it reads. Occasionally, we consider the computational complexity of the checker.

One could imagine using a checker in a communication system in which one has the option of requesting the retransmission of a message. A checker would be able to read only a constant number of bits of a received signal and then estimate the chance that a decoder will be able to correct its errors. If the checker determines that it is unlikely that the decoder will be able to correct the errors, then the checker can instantly request a retransmission of that block, before the decoder has wasted its time trying to decode the message. Unfortunately, all known codes with such checkers have rates that approach zero. Thus, current constructions would be inappropriate for use in real communication systems.

Usually, we will impose more structure on the bits of a codeword and the ways in which a checker can access them. We will partition the bits of a code into *segments* and insist that a checker read all the bits of a segment if it reads any one. For example, if s is a segment that contains the i-th through j-bits, and if $\vec{x} = (x_1, \ldots, x_n)$ is a word, then the value of x on segment s is $(x_i, \ldots, x_j)$. If an algorithm reads x_i, we will charge it for reading $x_{i+1}, \ldots, x_j$ as well. Note that we define segements to be *disjoint* sets of bit-positions. When we describe a checker of a code whose bits are broken into segments, we will count how many segments of the code a checker reads and how many bits those segments contain.

Remark 4.1.2 This division of the bits into segments can exist purely in the mind of the checker. It does not imply that any formatting symbols appear in the data. We describe the bits of a code as being broken into segments to indicate the way in which our algorithms will access the bits of that code.

The checkable codes in holographic proofs are derived from verifiable codes. In addition to being checkable, a verifiable code has the property that one can probabilistically verify that individual bits of a received word have not been corrupted.

Definition 4.1.3 Let $\{\mathcal{C}_i\}$ be a family of codes of lengths $\{n_i\}$ and relative minimum distance δ over an alphabet Σ such that the bits of each code are partitioned into segments. A *verifier* for $\{\mathcal{C}_i\}$ is a probabilistic algorithm that takes a word w and the name of a segment s as input such that

- if w is a word of $\mathcal{C}_i$, then the verifier accepts, and
- if the probability that the verifier accepts a word w (of length n_i) on segment s is greater than $1/2$, then there is a unique codeword c of $\mathcal{C}_i$ of relative minimum distance at most $\delta/3$ from w such that c has the same value as w on segment s.

As with checkable codes, we will measure the performance of a verifier by the number of segments of the code that it reads and by how many bits those segments contain.

We like to view the set of holographic proofs of a statement as being a checkable code with semantic restrictions.[2] Consider a code consisting of the proofs of a given length of a certain statement. If the statement is false or if it has no proofs of that length, then the code will be empty, and this is OK. Now, assume that this code also has a checker that reads only a constant number of bits of its input. The checker will accept every codeword in the code, and if it accepts some word with high probability, then that word must be close to a codeword. We have just described a holographic proof system.

Current constructions of holographic proofs have verifiers that read only a constant number of bits of their input. Thus, as one checks a holographic proof that a circuit is

[2] Consider a circuit that has many satisfying assignments. For each of these satisfying assignments, we obtain a different holographic proof that the circuit is satisfiable. We view this set of proofs as words in an error-correcting code.

satisfied, one can probabilistically obtain any bit of the satisfying assignment of the circuit while only reading a constant number of bits of the proof. This fact will be very important to us in Section 4.4.

4.2 Bi and Trivariate codes

A remarkable property of holographic proofs is that they enable us to demonstrate properties of some object while only examining a few bits of that object. In this section, we will take a first step in this direction by presenting a constant rate code of length n that can be checked and verified by examining $O(\sqrt{n})$ of its bits. Actually, the code that we develop will be somewhat stronger than this. The code will be divided into segments. Each segment will have size $O(\sqrt{n})$, and we will be able to check and verify the code while only examing data from a constant number of segments. We will later use these codes in a recursive construction to obtain codes that are checkable by examining fewer bits in more segments. Versions of these polynomial codes and their checking and verification algorithms first appeared in [AS92b] and [Sud92]. Our contribution is an improvement in their analysis (Theorem 4.2.19) which enables us to use versions of these codes that have much less redundancy than was previously possible.

The codes we construct in this section are polynomial codes over a field. We will use $\mathcal{F}$ to denote this field. It doesn't matter what field we use, so long as it is big enough to contain the objects that we describe.

The codewords in our first code will correspond to polynomials in two variables, say x and y, such that their degree in x is bounded by a parameter d and their degree in y is bounded by a parameter e. Since we will use such vector spaces often in this chapter, we will memorialize their description in a definition:

Definition 4.2.1 A polynomial $p(x_1, \ldots, x_k)$ has degree $(d_1, \ldots, d_k)$ if the degree of p in x_i is at most d_i, for each i. We do not demand that the d_i's be integers.

There are a number of ways that we could present a polynomial of degree (d, e). The most natural way of presenting such a polynomial would be to write down its coefficients (Figure 4-1a). Another way of presenting such a polynomial is to choose $X \subseteq \mathcal{F}$ and $Y \subseteq \mathcal{F}$ so that $|X| > d$ and $|Y| > e$, and then write down the values of p at each point of $X \times Y$ (Figure 4-1b) [3]. Both of these presentations involve writing down at least $(d+1)(e+1)$ pieces of information (elements of $\mathcal{F}$).

[3] The reader should verify that such a presentation uniquely defines the polynomial p.

Our presentations of polynomials will contain even more information. While the extra information will not be necessary to specify the polynomial, it will be useful when we want to establish facts about the polynomial without reading its entire presentation.

We now define a *presentation* of a polynomial to be the list of its values over some domain, along with the list of univariate polynomials obtained when one of the its variables is restricted to a value in that domain (Figure 4-1c).

Definition 4.2.2 Let $p(x, y)$ be a polynomial of degree (d, e) over a field $\mathcal{F}$, and let $X, Y \subseteq \mathcal{F}$. A *presentation of p over $X \times Y$* consists of

- the list of values of p at each point of $X \times Y$:

$$(p(x, y) : x \in X \text{ and } y \in Y), \text{ and}$$

- for each $x_0 \in X$, the coefficients of the univariate degree e polynomial obtained by restricting p to x_0 (we can view this as a function from X to the space of degree e polynomials in y), and
- for each $y_0 \in Y$, the coefficients of the univariate degree d polynomial obtained by restricting p to y_0 (which we can view as a function from Y to the space of degree d polynomials in x).

A presentation is viewed as being divided into segments. Each value of p and each univariate polynomial listed in the presentation is a separate segment. We refer to $X \times Y$ as the *domain* of the presentation.

Remark 4.2.3 When we write "list of values of p", we mean that the data should be organized into a list in which each element has the same length so that if one wants to look up the value of p at (x, y), then one knows instantly where to look in the list. The same should hold for the lists of univariate polynomials. Thus, these three lists could be viewed as one function from $X \times Y \rightarrow \mathcal{F}$ followed by a function from $X \rightarrow \mathcal{F}^{e+1}$ followed by a function from $Y \rightarrow \mathcal{F}^{d+1}$.

Remark 4.2.4 The definition of a presentation of a degree (d, e) polynomial naturally generalizes to degree $(d_1, \ldots, d_k)$ polynomials. A presentation of a degree $(d_1, \ldots, d_k)$ polynomial

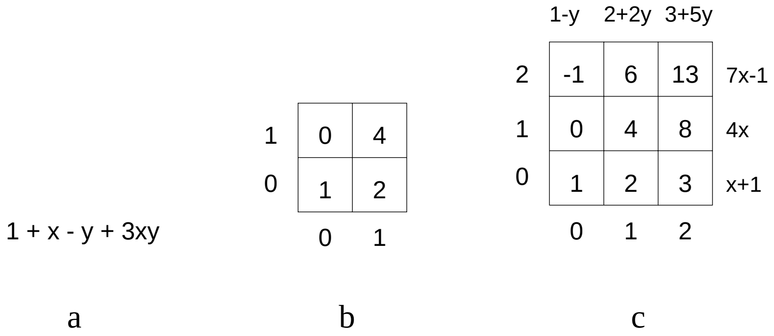

Figure 4-1: Three ways to describe a polynomial. c is a *presentation*

over a domain $X_1 \times \cdots \times X_k$ should consist of the list of values of the polynomial over that domain, and k lists of the univariate polynomials obtained by restricting the polynomial in all but one of its variables.

Presentations of distinct polynomials over sufficiently large domains are far apart. This follows easily from the following lemma by considering the polynomial which is their difference.

Lemma 4.2.5 [Schwartz [Sch80]] Let p be a degree $(d_1, \ldots, d_k)$ polynomial over a domain $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_k$ where $|\mathcal{D}_i| = n_i$ for $i = 1 \cdots k$. If p has more than

$$|\mathcal{D}| \sum_{i=1}^{k} \frac{d_i}{n_i}$$

zeros in $\mathcal{D}_1 \times \ldots \times \mathcal{D}_k$, then p must be the zero polynomial.

Proof: We prove this by induction. We already know the base case: a non-zero univariate polynomial of degree d can have at most d zeros. Now, assume that we have proved the lemma for $k-1$. There exist degree $(d_1, \ldots, d_{k-1})$ polynomials $p_0, \ldots, p_{d_k}$ such that

$$p(x_1, \ldots, x_k) = \sum_{i=0}^{d_k} p_i(x_1, \ldots, x_{k-1}) x_k^i .$$

If one of the p_i's is non-zero, then it is zero for at most a $\sum_{i=1}^{k-1} \frac{d_i}{n_i}$ of the values of $(x_1, \ldots, x_{k-1})$. For the values of $(x_1, \ldots, x_{k-1})$ such that it is non-zero, p can be zero at most a $\frac{d_k}{n_k}$ fraction of the time. ■

The first thing we will prove about presentations of bivariate polynomials is that if we are given some data and we are told that it is a presentation of a bivariate polynomial, then we can probablistically check whether it is a presentation of a bivariate polynomial while only reading a constant number of segments of the data. It is not possible to prove such a statement for presentations over domains that are too small, so our theorem will only apply to domains that are larger than the degree of the polynomial by some factor. Our contribution is that we prove this for domains whose size is greater than de by only a constant factor. Previously, Arora and Safra [AS92b] proved such a statement for domains whose size was cubic in de. Sudan [Sud92] later improved this to quadratic. In order to obtain nearly-linear size holographic proofs, we needed to improve their bounds.

In order to state the checking algorithm, we will need to state precisely what we mean by "data" that could be a presentation of a bivariate polynomial. The following definition essentially describes a string that is of the same length as a presentation of a degree (d, e) polynomial. The division of the string into segments is purely for the convenience of our description, and places no restriction on the definition.

Definition 4.2.6 A (d, e)-*presentation over* $X \times Y$ consists of:

- A string that we view as a list of elements of $\mathcal{F}$, one for each point of $X \times Y$; we say this string *assigns* an element of $\mathcal{F}$ to each point of $X \times Y$;
- for each $x_0 \in X$, a string whose length is the same as the length of the description of a degree e univariate polynomial over $\mathcal{F}$; and
- for each $y_0 \in Y$, a string whose length is the same as the length of the description of a degree d univariate polynomial over $\mathcal{F}$.

We will just write *presentation* if (d, e) is clear from context.

Remark 4.2.7 This definition does not actually impose any constraints on the content of the (d, e)-presentation. Rather, it should be viewed as a description of how the presentation is broken into segments and how those segments will be interpreted by the algorithms that read them.

Remark 4.2.8 This definition naturally generalizes to a definition of a $(d_1, \ldots, d_k)$-presentation in a fashion analogous to Remark 4.2.4.

We have defined a (d, e)-*presentation* so that a presentation of a degree (d, e) polynomial is one. We will now state the algorithm for checking whether a (d, e)-presentation is a presentation of a degree (d, e) polynomial.

Bivariate presentation checking algorithm:

- Choose a point (x, y) uniformly at random from $X \times Y$.
- Examine the value, v, assigned to that point, the string that is assigned to x, and the string assigned to y (in a presentation of a degree (d, e) polynomial, these represent the value of the polynomial at (x, y), its restriction to x, and its restriction to y). Accept if the string assigned to x represents a polynomial that takes the value v when evaluated at y and the string assigned to y represents a polynomial that takes the value v when evaluated at x.

The reader should verify that if this algorithm is run on a presentation of a bivariate degree (d, e) polynomial on $X \times Y$, then the algorithm will always accept. We will now see that the converse of this statement is true. That is, if this algorithm accepts with probability 1, then the presentation must be a presentation of a bivariate polynomial of degree (d, e).

Proposition 4.2.9 [Well-known] Let $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_n\}$, and let $d < m$ and $e < n$. Let $f(x, y)$ be a function on $X \times Y$ such that for $1 \leq j \leq n$, $f(x, y_j)$ agrees on X with some degree d polynomial in x, and for $1 \leq i \leq m$, $f(x_i, y)$ agrees on Y with some degree e polynomial in y. Then, there exists a polynomial $P(x, y)$ of degree (d, e) such that $f(x, y)$ agrees with $P(x, y)$ everywhere on $X \times Y$.

Proof: Recall that a degree d univariate polynomial is uniquely determined by its values at $d + 1$ points. For $1 \leq j \leq e + 1$, let $p_j(x)$ be the degree d polynomial that agrees with $f(x, y_j)$. For $1 \leq j \leq e + 1$, let $\delta_j(y)$ be the degree e polynomial in y such that

$$\delta_j(y_k) = \begin{cases} 1, & \text{if } j = k, \text{ and} \\ 0, & \text{if } 1 \leq k \leq e + 1, \text{ but } j \neq k. \end{cases}$$

We let $P(x,y) = \sum_{j=1}^{e+1} \delta_j(y) p_j(x)$. It is clear that P has degree (d,e). Moreover, $P(x, y_j) = f(x, y_j)$ for all $x \in X$ and $1 \leq j \leq d+1$. To see that in fact $P(x,y) = f(x,y)$ for all $(x,y) \in X \times Y$, observe that P and f agree at $e+1$ points in column y. Since f agrees with some degree e polynomial in column y, that polynomial must be the restriction of P to column y. ■

Sections 4.2.1 through 4.2.3 will be devoted to proving that, if the algorithm accepts a (d,e)-presentation with probability close to 1, then that presentation must look a lot like a presentation of some degree (d,e) polynomial. We formalize the notion of presentation being close to a presentation of a degree (d,e) polynomial by:

Definition 4.2.10 A (d,e)-presentation, P, over a domain $X \times Y$ is ϵ-*good* if there exists a degree (d,e)-polynomial p such that the presentation of p over $X \times Y$ differs from P in at most an ϵ fraction of their segments in each list.

We note that if a presentation P is ϵ-good for a sufficiently small ϵ, then there is a unique polynomial p to whose presentation P is close:

Proposition 4.2.11 Let P be a ϵ-good presentation of a degree $(d_1, \ldots, d_k)$ polynomial over a domain $\mathcal{D}_1 \times \ldots \times \mathcal{D}_k$ where $|\mathcal{D}_i| = n_i$, for $i = 1 \cdots k$. If

$$2 \sum_{i=1}^{k} \frac{d_i}{n_i} < \epsilon,$$

then there is a unique polynomial p such that P and the presentation of p differ in at most an ϵ fraction of their segments.

Proof: Assume, by way of contradiction, that there were two degree $(d_1, \ldots, d_k)$ polynomials p and q that agreed with P in all but an ϵ fraction of their segments. Then the presentations of p and q agree in all but at most a 2ϵ fraction of their segments. So, p and q must have the same values for at least a 2ϵ fraction of the domain. This would imply that $p - q$ is zero for a 2ϵ fraction of the domain. However, by Lemma 4.2.5, this would imply that $p - q \equiv 0$, contradicting the assumption that p and q were distinct. ■

Our aim is to now prove

Lemma 4.2.12 Let P be a (d,e) presentation over a domain $X \times Y$ such that $|X| > 8d$ and $|Y| > 8e$. If the probability that the bivariate presentation checking algorithm accepts is greater than $1-\epsilon$, for $\epsilon < 1/16$, then the presentation p is 3ϵ good. That is, the code of (d,e)-presentations over the domain $X \times Y$ has a checker that reads only a constant number of segments of its input, each of size $O(\sqrt{n})$.

Proof: The first part follows from Theorem 4.2.19, which is to be proved in Sections 4.2.1, 4.2.2, and 4.2.3. An alternative approach to proving this Theorem is outlined in Section 4.2.4. To obtain the checker, it is necessary to run the bivariate presentation checking algorithm a constant number of times. ■

4.2.1 The First Step

In our proof of Theorem 4.2.19, we will restrict our attention to the lists of univariate polynomials contained within a presentation. In Remark 4.2.3, we observed that the list of univariate polynomials of degree e in y could be viewed as a function from $X \rightarrow \mathcal{F}^{e+1}$. We can also view this list as a function in x and y that is arbitrary in the x direction, but always looks like a degree e polynomial in y. Because X has size m, an arbitrary function on X can be represented as a degree m polynomial.[4] Thus, we can also view this list as a description of a degree (m,e) polynomial in x and y. We will call this polynomial $C(x,y)$. We will obtain a degree (d,n) polynomial from the other list, and call that polynomial $R(x,y)$. If the bivariate presentation checking algorithm accepts with high probability then $R(x,y)$ and $C(x,y)$ agree on most of $X \times Y$.

Let $C(x,y)$ be a polynomial of degree (m,e) and let $R(x,y)$ be a polynomial of degree (d,n) such that

$$\operatorname*{Prob}_{(x,y)\in X\times Y}[R(x,y) \neq C(x,y)] \leq \gamma.$$

In Theorem 4.2.19, we will show that if γ is a sufficiently small constant, then there exists

[4] Actually, it can be represented as a degree $m-1$ polynomial. But, we will ignore this fact because it is a pain to write "$m-1$" everywhere.

a polynomial $Q(x, y)$ of degree (d, e) such that

$$\operatorname*{Prob}_{(x,y)\in X\times Y}[R(x,y) \neq Q(x,y) \text{ or } C(x,y) \neq Q(x,y)] \quad < \quad 2 \operatorname*{Prob}_{(x,y)\in X\times Y}[R(x,y) \neq C(x,y)].$$

As in [Sud92], we begin by finding a low-degree "error correcting" polynomial that is zero whenever R and C disagree.

Lemma 4.2.13 Let $S \subset X \times Y$ be a set of size at most $\delta^2 mn$. Then, there exists a non-zero polynomial $E(x, y)$ of degree $(\delta m, \delta n)$ such that $E(x, y) = 0$ for all $(x, y) \in S$.

Proof: The set of polynomials of degree $(\delta m, \delta n)$ is a vector space of dimension $(\lfloor \delta m \rfloor + 1)(\lfloor \delta n \rfloor + 1)$. Consider the map that sends a polynomial to the vector of values that it takes for each point in S. That is, let $S = \{s_1, \ldots, s_k\}$ and consider the map

$$\phi : E(x, y) \mapsto (E(s_1), E(s_2), \ldots, E(s_k)).$$

This map is a homomorphism of a vector space of dimension $(\lfloor \delta m \rfloor + 1)(\lfloor \delta n \rfloor + 1)$ into a vector space of dimension at most $\delta^2 mn$, which is smaller. So, there must be a non-zero polynomial in the vector space of polynomials of degree $(\delta m, \delta n)$ that evaluates to zero at every point in S. ■

Let S be the subset of $X \times Y$ on which R and C disagree. By Lemma 4.2.13, we can choose $E(x, y)$ so that

$$R(x, y)E(x, y) = C(x, y)E(x, y) \text{ for all } (x, y) \in X \times Y.$$

Moreover, $C(x, y)E(x, y)$ is a polynomial of degree $(m + \delta m, e + \delta n)$ and $R(x, y)E(x, y)$ is a polynomial of degree $(d + \delta m, n + \delta n)$. By Proposition 4.2.9, there exists a polynomial $P(x, y)$ of degree $(d + \delta m, e + \delta n)$ such that

$$R(x, y)E(x, y) = C(x, y)E(x, y) = P(x, y), \text{ for all } (x, y) \in X \times Y. \tag{4.1}$$

We would like to divide P by E as formal polynomials and conclude the proof. However, the most we can say is that

$$\frac{P(x,y)}{E(x,y)} = R(x,y) = C(x,y),$$

for all $(x,y) \in X \times Y$ such that $E(x,y) \neq 0$.

The next two sections will be devoted to showing that if n is sufficiently large, then E does in fact divide P. We will begin with one small step:

Lemma 4.2.14 Let $E(x,y)$, $P(x,y)$, $R(x,y)$ and $C(x,y)$ be polynomials of degrees $(\delta m, \delta n)$, $(d+\delta m, e+\delta n)$, (d,n) and (m,e) respectively such that (4.1) holds. If $|X| > \delta m + d$ and $|Y| > \delta n + e$, then for all $y_0 \in Y$ and for all $x_0 \in X$, $P(x,y_0) \equiv R(x,y_0)E(x,y_0)$ and $P(x_0,y) \equiv C(x_0,y)E(x_0,y)$.

Proof: For fixed y_0, $P(x,y_0)$ and $R(x,y_0)E(x,y_0)$ are degree $d+\delta m$ polynomials that have the same value on at least $d+\delta m+1$ points, so $P(x,y_0) \equiv R(x,y_0)E(x,y_0)$ as formal polynomials in x. The other case is proved similarly. ■

From this point, we know two routes to the proof of Lemma 4.2.12. The route that we pursue in the main text (Sections 4.2.2 and 4.2.3) is more elementary. In Section 4.2.4, we show how one can replace Section 4.2.2 and Lemma 4.2.18 with an application of Bezout's Theorem. This route is more direct, but may be accessible to fewer readers.

4.2.2 Resultants

In this section, we will review some standard facts about resultants. A more complete presentation can be found in [Lan93, vdW53]. We note that Sudan [Sud92] introduced the idea of using the resultant to prove that E divides P.

Let $\mathcal{F}$ be a field and let

$$\begin{aligned} P(x) &= P_0 + P_1 x + \cdots + P_r x^r, \qquad \text{and} \\ E(x) &= E_0 + E_1 x + \cdots + E_s x^s \end{aligned}$$

be polynomials in x with coefficients in the field $\mathcal{F}$.

Proposition 4.2.15 $P(x)$ and $E(x)$ have a non-trivial common factor if and only if there exist polynomials $A(x)$ of degree $s-1$ and $B(x)$ of degree $r-1$ such that $P(x)A(x)-E(x)B(x)=0$.

Proof: If there exist $F(x)$, $\hat{P}(x)$ and $\hat{E}(x)$ such that $\deg F(x) \geq 1$, $P(x) = F(x)\hat{P}(x)$, and $E(x) = F(x)\hat{E}(x)$, then we can choose $A(x) = \hat{E}(x)$ and $B(x) = \hat{P}(x)$.

To go the other direction, assume that such $A(x)$ and $B(x)$ exist. Since the degree of $P(x)$ is greater than the degree of $B(x)$, $P(x)$ and $E(x)$ must share a common factor. ■

We can reformulate this as a system of linear equations in the coefficients of A and B:

$$\begin{array}{llllllll}
P_0A_0 & & & = & E_0B_0 & & & \\
P_1A_0+ & P_0A_1 & & = & E_1B_0+ & E_0B_1 & & \\
P_2A_0+ & P_1A_1+ & P_0A_2 & = & E_2B_0+ & E_1B_1+ & E_0B_2 & \\
 & & \ddots & \vdots & & & & \ddots \\
 & & P_rA_{s-1} & = & & & & E_sB_{r-1}
\end{array}$$

If we treat the coefficients of A and B as the variables of a system of linear equations, then we find that the above equations have a solution if and only if the matrix $M(P,E)$ has determinant zero, where $M(P,E) =$

$$\left(\begin{array}{cccccccc}
P_r & P_{r-1} & \dots & \dots & P_0 & 0 & \dots & 0 \\
0 & P_r & & \dots & P_1 & P_0 & \dots & 0 \\
\vdots & \ddots & \ddots & & & \ddots & \ddots & \vdots \\
0 & \dots & 0 & P_r & \dots & \dots & P_1 & P_0 \\
E_s & E_{s-1} & \cdots & E_0 & 0 & \dots & \dots & 0 \\
\vdots & \ddots & & & & \ddots & & \vdots \\
\vdots & & \ddots & & & & \ddots & 0 \\
0 & \dots & & 0 & E_s & E_{s-1} & \dots & E_0
\end{array}\right)
\begin{array}{l}
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\end{array}}\right\} s \text{ rows} \\
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\end{array}}\right\} r \text{ rows}
\end{array}$$

We now define $R(P,E)$, the resultant of P and E, to be the polynomial in the coefficients of P and E obtained by taking the determinant of $M(P,E)$. We obtain:

Proposition 4.2.16 The polynomials $P(x)$ and $E(x)$ share a common factor if and only if $R(P,E)=0$.

The following fact about the derivative of the determinant of a matrix of polynomials will play a crucial role in our proof: Let

$$M(x) = \begin{pmatrix} p_{1,1}(x) & p_{1,2}(x) & \dots & p_{1,k}(x) \\ p_{2,1}(x) & p_{2,2}(x) & \dots & p_{2,k}(x) \\ \vdots & \vdots & \ddots & \vdots \\ p_{k,1}(x) & p_{k,2}(x) & \dots & p_{k,k}(x) \end{pmatrix}$$

be a k-by-k matrix of polynomials in x over $\mathcal{F}$ and let $R(x)$ be the determinant of $M(x)$.

Proposition 4.2.17 $R'(x)$, the derivative of $R(x)$, can be expressed as

$$R'(x) = \begin{vmatrix} p'_{1,1}(x) & p'_{1,2}(x) & \dots & p'_{1,k}(x) \\ p_{2,1}(x) & p_{2,2}(x) & \dots & p_{2,k}(x) \\ \vdots & \vdots & \ddots & \vdots \\ p_{k,1}(x) & p_{k,2}(x) & \dots & p_{k,k}(x) \end{vmatrix} + \cdots + \begin{vmatrix} p_{1,1}(x) & p_{1,2}(x) & \dots & p_{1,k}(x) \\ p_{2,1}(x) & p_{2,2}(x) & \dots & p_{2,k}(x) \\ \vdots & \vdots & \ddots & \vdots \\ p'_{k,1}(x) & p'_{k,2}(x) & \dots & p'_{k,k}(x) \end{vmatrix}$$

4.2.3 Presentation checking theorems

Since the propositions of the previous section concerned univariate polynomials, you may be wondering how we are going to apply them to bivariate polynomials. The idea is to treat the polynomials $P(x, y)$ and $E(x, y)$ as polynomials in y over $\mathcal{F}(x)$, the field of rational functions in x. $\mathcal{F}(x)$ is the field comprising terms of the form $p(x)/q(x)$, where $p(x)$ and $q(x)$ are polynomials in $\mathcal{F}$. It is easy to verify that this is in fact a field.

We can now consider P and E as polynomials in y with coefficients in $\mathcal{F}(x)$ by writing

$$\begin{aligned} P(x, y) &= P_0(x) + P_1(x)y + \cdots + P_{\delta n+e}(x)y^{\delta n+e} \\ E(x, y) &= E_0(x) + E_1(x)y + \cdots + E_{\delta n}(x)y^{\delta n}. \end{aligned}$$

We will show that E divides P as a polynomial in y over the field $\mathcal{F}(x)$. By Gauss' Lemma[5], this implies that E divides P over $\mathcal{F}[x]$, the ring of *polynomials* in x, which means that $E(x, y)$ divides $P(x, y)$.

[5] The usual statement of Gauss' Lemma is that if a polynomial with integer coefficients can be factored over the rationals, then it can be factored over the integers. A proof of Gauss' Lemma can be found in most undergraduate algebra textbooks.

We will begin our proof by dividing E and P by their greatest common divisor. If that greatest common divisor is not E, then we obtain two polynomials with no common factor. To obtain a contradiction, we will show that these two polynomials have a common factor when considered as polynomials in y over $\mathcal{F}(x)$. By Gauss' Lemma, this will imply that they share a common factor when considered as polynomials in x and y.

Lemma 4.2.18 Let $E(x, y)$ be a polynomial of degree $(\alpha m, \beta n)$ and let $P(x, y)$ be a polynomial of degree $(\alpha m + \delta m, \beta n + \epsilon n)$. If there exist distinct $x_1, \ldots, x_m$ such that $E(x_i, y)$ divides $P(x_i, y)$ for $1 \leq i \leq m$, distinct $y_1, \ldots, y_n$ such that $E(x, y_i)$ divides $P(x, y_i)$ for $1 \leq i \leq n$ and if

$$1 > \alpha + \beta + \delta + \epsilon,$$

then $E(x, y)$ divides $P(x, y)$.

Proof: Assume, without loss of generality, that $\beta \geq \alpha$. Let $F(x, y)$ be the largest common factor of $P(x, y)$ and $E(x, y)$. Assume by way of contradiction that $F \not\equiv E$ and that $F(x, y)$ has degree (a, b). Set

$$P(x, y) \equiv \hat{P}(x, y) F(x, y) \quad \text{and} \quad E(x, y) \equiv \hat{E}(x, y) F(x, y).$$

We will now divide P and E by F and apply the lemma to $\hat{P}$ and $\hat{E}$ over the rows and columns on which F is not identically zero. The conditions of the lemma are satisfied by $\hat{P}$ and $\hat{E}$ on this domain because F can be identically zero on at most b rows and a columns, and

$$\alpha + \beta + \delta + \epsilon \geq \frac{\alpha m - a}{m - a} + \frac{\delta m - a}{m - a} + \frac{\beta n - b}{n - b} + \frac{\epsilon n - b}{n - b}.$$

Thus, we can assume without loss of generality that $P(x, y)$ and $E(x, y)$ have no common factors. We will use this assumption to obtain a contradiction. Write

$$\begin{aligned} P(x, y) &\equiv P_0(x) + P_1(x) y + \cdots + P_{\beta n + \epsilon n}(x) y^{(\beta + \epsilon) n} \\ E(x, y) &\equiv E_0(x) + E_1(x) y + \cdots + E_{\beta n}(x) y^{\beta n}, \end{aligned}$$

and form the matrix $M(P, E)(x) =$

$$\left(\begin{array}{cccccc} P_{(\beta+\epsilon)n}(x) & \dots & \dots & P_0(x) & \dots & 0 \\ \vdots & \ddots & & & \ddots & \vdots \\ 0 & \dots & P_{(\beta+\epsilon)n}(x) & \dots & \dots & P_0(x) \\ E_{\beta n}(x) & \cdots & E_0(x) & 0 & \dots & 0 \\ \vdots & \ddots & & & \ddots & 0 \\ 0 & \dots & 0 & E_{\beta n}(x) & \dots & E_0(x) \end{array}\right) \begin{array}{l} \left.\vphantom{\begin{array}{c} P \\ \vdots \\ P \end{array}}\right\} \beta n \\ \left.\vphantom{\begin{array}{c} E \\ \vdots \\ E \end{array}}\right\} (\beta+\epsilon)n \end{array}$$

$R(P, E)(x)$, the resultant of P and E, is the determinant of $M(P, E)(x)$ and can therefore be viewed as a polynomial in x. $M(P, E)(x)$ has βn rows of coefficients of P and $(\beta+\epsilon)n$ rows of coefficients of E, so $R(P, E)(x)$ will be a polynomial of degree at most $mn(\beta(\alpha+\delta)+(\beta+\epsilon)\alpha)$. We will show that $R(P, E)(x)$ is in fact the zero polynomial by demonstrating that it has more than this many roots.

For $1 \leq i \leq n$, $E(x_i, y)$ divides $P(x_i, y)$, so we can see that the first βn rows of $M(P, E)(x_i)$ are dependent on the last $(\beta + \epsilon)n$ rows of $M(P, E)(x_i)$. This implies that $M(P, E)(x_i)$ is a matrix of rank at most $(\beta + \epsilon)n$ (actually, the rank is exactly $(\beta + \epsilon)n$). By Proposition 4.2.17, the k-th derivative of $R(P, E)(x)$ at x_i is the sum of determinants of matrices of rank at most $(\beta + \epsilon)n + k$. Since $M(P, E)(x)$ is a matrix of side $(2\beta + \epsilon)n$, $R^{(k)}(P, E)(x_i)$ is zero for $k < \beta n$. That is, $R(P, E)(x)$ has a zero of multiplicity βn at each of $x_1, \dots, x_m$. Because we assumed that $1 > \alpha + \beta + \delta + \epsilon$ and $\beta \geq \alpha$, we find

$$m(\beta n) > mn(\beta\alpha + \beta\delta + \beta^2 + \beta\epsilon) \geq mn(\beta\alpha + \beta\delta + \alpha\beta + \alpha\epsilon),$$

so $R(P, E)(x)$ must be the zero polynomial. Applying Proposition 4.2.16, we see that E and P must have a non-trivial common factor when considered as polynomials in y over $\mathcal{F}(x)$, which is a contradiction. ■

We can now prove:

Theorem 4.2.19 [Bivariate Checking] Let $\mathcal{F}$ be a field, let $X = \{x_1, \dots, x_m\} \subseteq \mathcal{F}$, and let $Y = \{y_1, \dots, y_n\} \subseteq \mathcal{F}$. Let $R(x, y)$ be a polynomial over $\mathcal{F}$ of degree (d, n) and let $C(x, y)$ be a polynomial over $\mathcal{F}$ of degree (m, e). If

$$\operatorname*{Prob}_{(x,y)\in X\times Y}[R(x,y) \neq C(x,y)] < \delta^2 \text{ and}$$

$$1 > 2\frac{d}{m} + 2\frac{e}{n} + 2\delta,$$

then there exists a polynomial $Q(x, y)$ of degree (d, e) such that

$$\operatorname*{Prob}_{(x,y)\in X\times Y}[R(x,y) \neq Q(x,y) \text{ or } C(x,y) \neq Q(x,y)] < 2\delta^2,$$

$$\begin{aligned}
\operatorname*{Prob}_{x_0\in X}[C(x_0,y) \equiv_y Q(x_0,y)] &> 1-2\delta^2, \text{ and} \\
\operatorname*{Prob}_{y_0\in Y}[R(x,y_0) \equiv_x Q(x,y_0)] &> 1-2\delta^2.
\end{aligned}$$

Proof: Let S be the set of points such that $R(x,y) \neq C(x,y)$. By Lemma 4.2.13, there exists a polynomial $E(x,y)$ of degree $(\delta m, \delta n)$ such that S is contained in the zero set of E. By Lemmas 4.2.14 and 4.2.18, there exists a polynomial $Q(x,y)$ of degree (d,e) such that

$$R(x,y)E(x,y) = C(x,y)E(x,y) = Q(x,y)E(x,y),$$

for all $(x,y) \in X \times Y$.

This implies that in any row on which $E(x,y)$ is non-zero, Q agrees with R on that entire row. However, E can be identically zero on at most δn rows; so, E must be non-zero on at least $(1-\delta)n$ rows. Thus, Q must agree with R on at least $(1-\delta)n$ rows. We can similarly show that Q must agree with C on at least $(1-\delta)m$ columns. We could stop now, content in the knowledge that R and C agree on the intersection of $(1-\delta)n$ rows and $(1-\delta)m$ columns; however, we will show that they agree on many more points.

As before, let S be the set of points at which $R(x,y) \neq C(x,y)$. Let T be the set of points at which $R(x,y) = C(x,y)$, but $Q(x,y) \neq R(x,y)$. We will show that $|T| \leq |S|$, which will prove the first inequality. Call the rows on which R disagrees with Q *bad* and define *bad* columns similarly. Let b_r be the number of bad rows and let b_c be the number of bad columns. Call *good* any row or column that is not bad. We will say that a row and column *disagree* if R and C take different values at their intersection. We first observe that there can be at most $e+b_r$ points of T in any bad column: if a column has more than $e+b_r$ points of T, then it must have at least $e+1$ points in good rows at which C agrees with R and therefore Q, implying that that column is in fact good. Because we assumed that $1 > 2\delta + e/n$, every bad column must have at least $n/2$ points of S in the intersection of that column with the good rows. We can similarly analyze the bad rows to see that each must have at least $m/2$ points of S in the intersection of that row with the good columns. We thereby conclude that $|T| \leq |S|$. (the basic idea is that the points of T must lie in the shaded region of Figure 4-2). By examining the two regions in Figure 4-2 that contain the points of S, we conclude

$$\frac{m}{2} b_r < \delta^2 nm \Rightarrow \frac{b_r}{n} < 2\delta^2, \quad \text{and} \quad \frac{n}{2} b_c < \delta^2 nm \Rightarrow \frac{b_c}{m} < 2\delta^2.$$

■

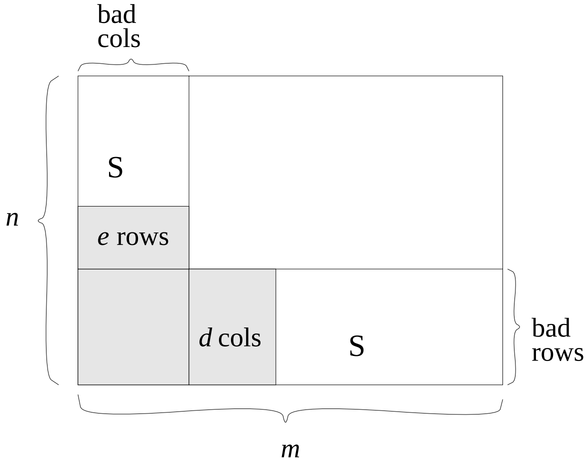

Figure 4-2: The arrangement of bad rows and columns.

When we construct holographic proofs, we will need a version of Lemma 4.2.12 that applies to trivariate presentations. We will not attempt to optimize this version.

While the following proof might look a little long, it is actually very simple: we will first apply Theorem 4.2.19 in each (w, x) plane and each (w, y) plane to obtain a collection of bivariate polynomials in (w, x) and (w, y) respectively. We will then treat these as univariate polynomials in x and y that assume values that are functions in w, and then combine them with another application of Theorem 4.2.19.

Corollary 4.2.20 Let $P(w, x, y)$, $R(w, x, y)$, and $C(w, x, y)$ be polynomials over a field $\mathcal{F}$ of degrees $(c, 12d, 12e)$, $(12c, d, 12e)$ and $(12c, 12d, e)$ respectively. Let W, X, and Y be subsets of $\mathcal{F}$ of sizes $12c$, $12d$, and $12e$ respectively. If

$$\operatorname*{Prob}_{(w,x,y)\in W\times X\times Y}[P(w, x, y) = R(w, x, y) = C(w, x, y)] > 1 - \frac{\epsilon}{2916},$$

for $\epsilon < 1$, then there exists a polynomial $Q(w, x, y)$ of degree (c, d, e) such that

$$\operatorname*{Prob}_{(w,x,y)\in W\times X\times Y}[Q(w, x, y) = P(w, x, y) = R(w, x, y) = C(w, x, y)] > 1 - \frac{2\sqrt{\epsilon}}{9}.$$

Proof: We begin by observing that there exists a set $Y' \subset Y$ such that $|Y'| \geq (1-\frac{\sqrt{\epsilon}}{54})\,|Y|$ and for each $y_0 \in Y'$,

$$\operatorname*{Prob}_{(w,x)\in W\times X}[P(w, x, y_0) = R(w, x, y_0)] > 1 - \frac{\sqrt{\epsilon}}{54}.$$

For each $y_0 \in Y'$, we now apply Theorem 4.2.19 to the polynomials obtained by restricting P and R to y_0 to obtain a degree (c, d) polynomial in w and x, which we will call R^{y_0}, such that

$$\operatorname*{Prob}_{x_0\in X}[R^{y_0}(x_0, w) \equiv_w P(w, x_0, y_0)] > 1 - \frac{2\sqrt{\epsilon}}{54}.$$

For $y_0 \notin Y'$, we let $R^{y_0} \equiv 0$. We are now going to treat the R^{y_0}'s as univariate polynomials in x that take on values that are functions of w. We then combine these together to form a degree (d, n) polynomial over $\mathcal{F}(w)$ that we will call R^Y by defining:

$$R^Y(x_0, y_0) = R^{y_0}(w, x_0).$$

We see that

$$\operatorname*{Prob}_{(x,y)\in X\times Y}[R^Y(x, y) \equiv_w P(w, x, y)] > 1 - \frac{3\sqrt{\epsilon}}{54}.$$

We now combine C and P in the same way to obtain a degree (m, e) polynomial over $\mathcal{F}(w)$ that we call C^X such that

$$\operatorname*{Prob}_{(x,y)\in X\times Y}[C^X(x,y) \equiv_w P(w,x,y)] > 1 - \frac{3\sqrt{\epsilon}}{54}.$$

By combining these two inequalities, we find

$$\operatorname*{Prob}_{(x,y)\in X\times Y}[C^X(x,y) \equiv_w R^Y(x,y) \equiv_w P(w,x,y)] > 1 - \frac{\sqrt{\epsilon}}{9}.$$

Since

$$1 > 2\frac{d}{12d} + 2\frac{e}{12e} + 2\sqrt{\frac{\sqrt{\epsilon}}{9}},$$

we can apply Theorem 4.2.19 to the polynomials R^Y and C^X to obtain a degree (d, e) polynomial $Q(x,y)$ that takes values in $\mathcal{F}(w)$ such that

$$\operatorname*{Prob}_{(x,y)\in X\times Y}[Q(x,y) \equiv_w C^X(x,y) \equiv_w R^Y(x,y) \equiv_w P(w,x,y)] > 1 - \frac{2\sqrt{\epsilon}}{9}.$$

It remains to show that $Q(x,y)$ is a degree c polynomial in w as well. To do this, consider a value $y_0 \in Y$ such that $Q(x,y)$ agrees with $P(w,x,y)$ for more than d values of x. On these values of x, $Q(x,y_0)$ is a degree d polynomial in w. Moreover, for any other value of x, the value of $Q(x,y_0)$ can be expressed as a linear combination with coefficients in $\mathcal{F}$ of these polynomials, so $Q(x,y_0)$ will be a degree d polynomial in w for all $x \in X$. We now observe by simple counting that there must be more than e values of y_0 for which this is true, so we can apply the same reasoning for these values of y_0, and then in the x direction, to see that $Q(x,y)$ is a degree (c,d,e) polynomial in w, x and y. ■

This corollary implies that there is a trivariate presentation checking algorithm analogous to the bivariate presentation checking algorithm.

4.2.4 Using Bezout's Theorem

In this Section, we describe a different way of proving Theorem 4.2.19. We use Bezout's Theorem which, roughly stated, says:

Theorem 4.2.21 [Bezout] Let $p(x,y)$ and $q(x,y)$ be polynomials of degree d and degree e respectively. If p and q have more than de zeroes in common (counting zeroes by multiplicity), then p and q share a common factor.

We now pick up our proof where we left off at the end of Section 4.2.1. For simplicity of exposition, we will only consider the case in which $n = m$ and $d = e$. The cases in which they are different are proved similarly.

Proposition 4.2.22 Let $E(x,y)$ and $P(x,y)$ be degree $(\delta n, \delta n)$ and degree $(d+\delta n, d+\delta n)$ polynomials over a field $\mathcal{F}$ such that $E(x_0, y)$ divides $P(x_0, y)$ as a polynomial in y for $5(d+\delta n)$ distinct values of x_0. Then, $E(x,y)$ divides $P(x,y)$ as a polynomial in x and y.

Proof: Let $x_1, \ldots, x_{5(d+\delta n)}$ be distinct values such that $E(x_i, y)$ divides $P(x_i, y)$ as a polynomial in y. Assume that E has degree exactly δn in y (it is to our advantage if this is not the case) and assume that $E(x_i, y)$ has full degree in y for $1 \leq i \leq 5d + 4\delta n$. Because $E(x_i, y)$ divides $P(x_i, y)$, $E(x,y)$ and $P(x,y)$ share at least δn zeroes[6] (counted by multiplicities) along the line $x = x_i$, for each $1 \leq i \leq 5d + 4\delta n$. Because

$$\deg(E) \cdot \deg(P) \leq 4\delta n \, (d + \delta n) < \delta n \, (5d + 4\delta n) \, ,$$

Bezout's Theorem implies that P and E share a common factor. If we divide E and P by this common factor and again apply the same argument, we eventually discover that E divides P as a polynomial in x and y. ■

One can essentially replace Section 4.2.2 and Lemma 4.2.18 with an unbalanced version of Proposition 4.2.22. The bounds that we thereby obtain in Theorem 4.2.19 are slightly weaker, but still sufficient for the purposes of this chapter.

4.2.5 Sub-presentations and verification

It is clear that a presentation of a polynomial on a domain $\mathcal{D}$ contains presentations of that polynomial on subsets of that domain. In this section, we will explain how to deal with

[6] For simplicity, we ignore the x_i's for which E does not have full degree. However, if we look back at Lemma 4.2.14, we see that E and P share zeroes at infinity at these x_i's.

these sub-presentations. We begin with the following simple observation:

Proposition 4.2.23 Let P be a ϵ-good (d_x, d_y, d_z)-presentation over a domain $\mathcal{D} = X \times Y \times Z$, and let $\mathcal{D}' = X' \times Y' \times Z'$ be a subset of $\mathcal{D}$ such that

$$c\,|\mathcal{D}'| \geq |\mathcal{D}|\,.$$

If we let P' consist of the elements of P that correspond to a presentation over $\mathcal{D}'$, then P' is $c\epsilon$-good. Moreover, if

$$\frac{d_x}{|X|} + \frac{d_y}{|Y|} + \frac{d_z}{|Z|} + 4c\epsilon < 1,$$

then the degree (d_x, d_y, d_z) polynomial to which P' is close is the same polynomial which P is close to.

Proof: The first part is obvious. The second follows from Lemma 4.2.5. ■

Tri-variate verification algorithm (at (x_0, y_0, z_0))

Remark: Assume that this algorithm is given a presentation P over $X \times Y \times Z$.

Remark: We will let ϵ be some small constant.

1. Check that the presentation P is ϵ-good.

2. Check that the sub-presentation on $X \times Y \times z_0$ is ϵ-good using the bivariate presentation checking algorithm.

3. Choose a constant number of points of $X \times Y \times z_0$ at random and check that the univariate polynomial of P in z that goes through each takes the value assigned to that point by the presentation.

4. Choose a constant number of points of $X \times y_0 \times z_0$ and check that the univariate polynomial of P in x that goes through each point takes the value assigned to that point by the presentation.

5. Check that the univariate polynomial of P in x that goes through (x_0, y_0, z_0) agrees with the value assigned to that point by P.

Another type of sub-presentation is one of lower dimension. For example, a trivariate presentation P over $X \times Y \times Z$ contains a bivariate presentation over $X \times Y \times z_0$, for $z_0 \in Z$. However, the fact that P is ϵ-good tells us little about this sub-presentation.

To show that the sub-presentation can be certified as good, we will show a stronger property of presentations: they have simple verifiers. That is, once we know that a presentation is ϵ-good for a sufficiently small ϵ, we can check whether any particular segment of the presentation is the same as the segment of the presentation of the polynomial that our presentation is close to.

Lemma 4.2.24 There exists a constant c such that for all $\epsilon > 0$ there exist constants in the trivariate verification algorithm such that if P is a (d_x, d_y, d_z)-presentation over a domain $X \times Y \times Z$ where

$$cd_x < |X|, \quad cd_y < |Y|, \text{ and } cd_z < |Z|, \text{ then}$$

- If the verification algorithm passes Step 1 with probability at least $1/2$, then there is a polynomial $p(x,y,z)$ of degree $(\cdot,\cdot,\cdot)$ such that P is ϵ-close to a presentation of $p(x,y,z)$.
- If the verification algorithm passes Steps 1, 2, and 3, with probability at least $1/2$, then the sub-presentation of P on $X \times Y \times z_0$ is ϵ-close to a presentation of $p(x,y,z_0)$.
- If the verification algorithm passes Steps 1 through 4 with probability at least $1/2$, then the univariate polynomial of P in x through $(\cdot, y_0, z_0)$ is $p(x, y_0, z_0)$.
- If the verification algorithm passes Steps 1 through 5 with probability at least $1/2$, then the value that P assigns to the point (x_0, y_0, z_0) is $p(x_0, y_0, z_0)$.

Proof: These facts follow from Lemma 4.2.5, Lemma 4.2.12 and Corollary 4.2.20. ■

This verification algorithm appears in [Sud92]. Our contribution is the realization that it works over domains whose size is linear in the degree of the presentations.

4.3 A simple holographic proof system

4.3.1 Choosing a key problem

We will base our holographic proof system on a particular NP-complete problem. We want to choose a problem such that

- the descriptions of instances of problems like 3SAT and circuit satisfiability can be easily and efficiently transformed into instances of our problem, and
- it is particularly easy to create a holographic proof system for our problem.

So that we can better understand the second objective, let us examine why it might be difficult to create a holographic proof system directly for the problem of circuit satisfiability. The usual way of describing an instance of circuit satisfiability is to describe a circuit by providing, for each gate in the circuit, its function and the names of its inputs. Checking whether an assignment satisfies a circuit can be a very irregular process: to compute the value of a gate in the circuit, one must first look up the names of the inputs to the gate and then their values. These gates could be anywhere in the circuit. When we try to design a holographic proof system directly for the circuit satisfiability problem, we find that the majority of our proof is devoted to making sure that the values of the gates are being shuttled around the proof correctly. The overhead that we thereby incur prevents us from finding an efficient holographic proof system for this problem.

We will design a problem such that when we want to determine whether one of its constraints is satisfied, we will know instantly where we should find the values that we need to examine. In our problem, one will be able to compute the names of the variables involved in a constraint just from the name of the constraint. We thereby eliminate the overhead associated with the irregular structure of problems like circuit satisfiability. Moreover, it will be easy to reduce problems such as circuit satisfiability to instances of our problem. So that we can apply the tools of Section 4.2, we will need to make our problem simple in an algebraic sense, but we will discuss this later.

The problem that we will use will be a coloring problem on a “wrapped de Bruijn graph”. An instance of the problem will consist of an assignment of a first color to each node in the graph. A solution of the problem will be an assignment of a second color to each node in the graph so that the pairs of colors assigned to each node and its neighbors satisfy certain coloring rules. Each coloring rule will restrict the configuration of colors that can be assigned to a node and its neighbors. The same rules will apply to every node in the graph. We will call an instance of the coloring problem a *coloring problem instance* and a solution of such an instance a *coloring problem solution.*

Let us recall the definition of a de Bruijn graph (see Figure 4-3 for an example).

Definition 4.3.1 The *de Bruijn graph B_n* is a directed graph on 2^n nodes in which each node is represented by an n-digit binary string. The node represented by the string $(x_1, \ldots, x_n)$ has edges pointing to the nodes represented by

$$(x_2, \ldots, x_n, x_1) \text{ and } (x_2, \ldots, x_n, (x_1 \oplus 1)),$$

where by $a \oplus b$ we mean the sum of a and b modulo 2.

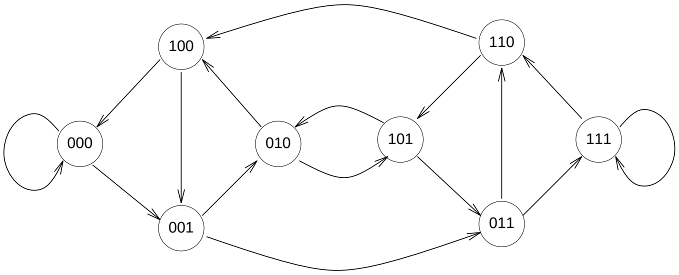

Figure 4-3: A de Bruijn graph

We will define a *wrapped de Bruijn graph* to be the product of a de Bruijn graph with a cycle, where in the product we will put an edge from vertex (x, a) to vertex (y, b) if and only if there are edges from x to y and a to b (See Figure 4-4). The size of the cycle needs to be some constant times n that is large enough for us to perform certain routing operations on the graph. $5n$ will suffice.

Definition 4.3.2 The *wrapped de Bruijn graph B_n* is a directed graph on $5n \cdot 2^n$ nodes in which each node is represented by a pair consisting of a number modulo $5n$ and an n-digit binary string. The node represented by the pair $(a, (x_1, \ldots, x_n))$ has edges pointing to the nodes represented by

$$(a+1,(x_2,\ldots,x_n,x_1)) \text{ and } (a+1,(x_2,\ldots,x_n,(x_1\oplus 1))),$$

where the addition $a+1$ is taken modulo $5n$. For a fixed $a \in \{0,\ldots,5n-1\}$, we will refer to the set of nodes whose representation begins with a as a *column* of the graph (to agree with a column in Figure 4-4).

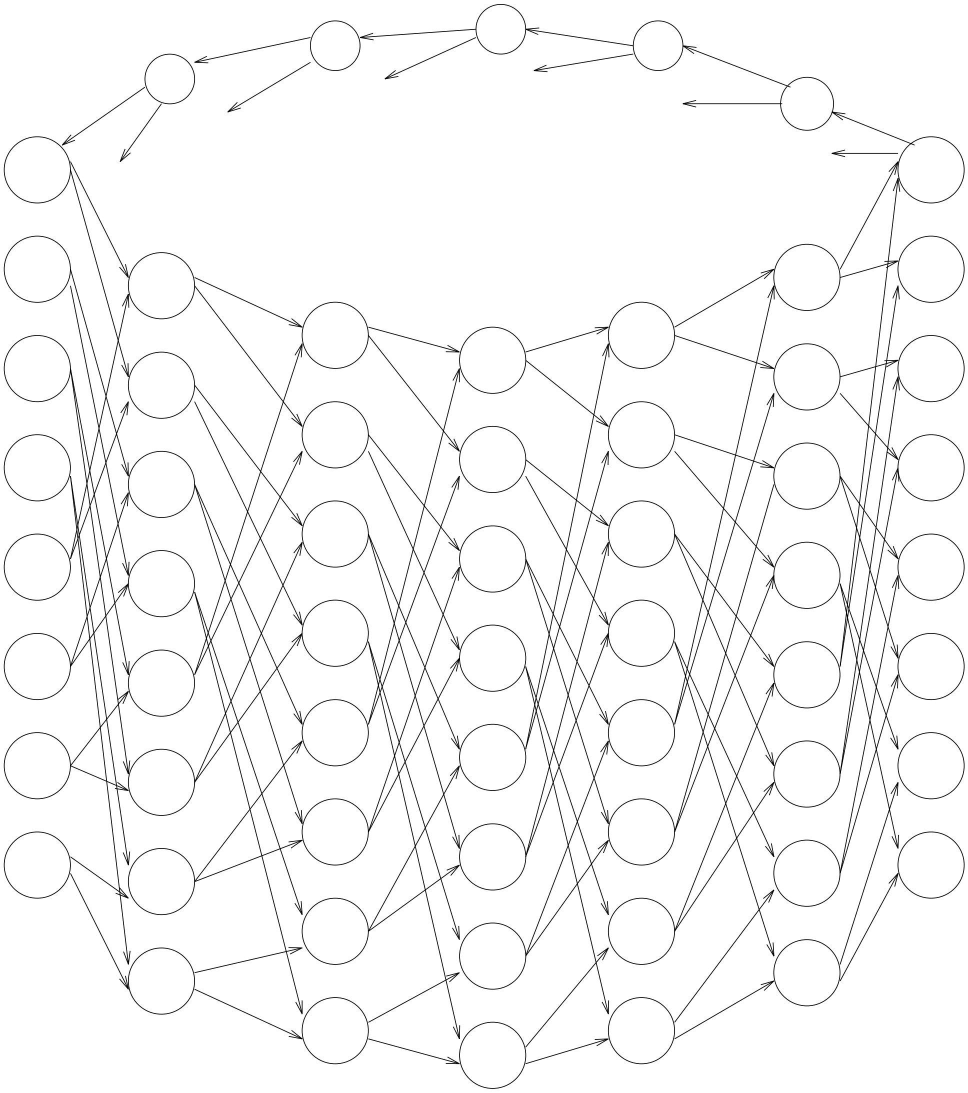

Figure 4-4: A wrapped de Bruijn graph

We use wrapped de Bruijn graphs for two reasons: one can route any permutation on such a graph and, as we will see in Section 4.3.2, they have a very simple algebraic description.

Our ability to route permutations on a wrapped de Bruijn graph enables us to "draw" a circuit on one. We begin by considering an ordinary drawing of a circuit (Figure 4-5). To draw the circuit on a wrapped de Bruijn graph, we will associate one column of the wrapped de Bruijn graph with the gates of the circuit (some nodes of the column may not be associated with any gate in the circuit). To each type of node that could appear in a circuit, we will assign a color; thus, the association of nodes in the circuit with nodes in the graph will be accomplished by coloring the nodes in the graph with the correct colors. For the convenience of later arguments, we will also create a special "output" node.

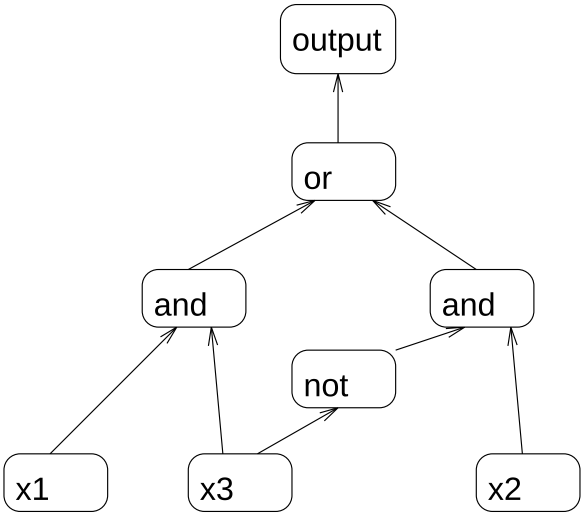

Figure 4-5: A drawing of a circuit

We now color the remaining nodes of the graph in a manner that describes the connections that appear between the gates in the circuit. To each of these remaining nodes, we associate a switching action such as one of those depicted in Figure 4-6 (again, by coloring a node with a color associated with its switching action). Note that we allow a switch to copy an incoming message and send it to both of its outputs (*e.g.* the switch in the lower left-hand corner of Figure 4-6). We view the task of assigning the switching actions that connect gates to their inputs as a routing problem in which each node has at most two incoming packets. By using standard packet-routing techniques (see [Lei92]), we see that $5n$ steps through a de Bruijn graph of 2^n nodes are sufficient to solve the routing problem. Thus, we can find switching actions for each of the nodes in the wrapped de Bruijn graph so that the output of each gate is routed to the inputs of those gates that need it (see Figure 4-7).

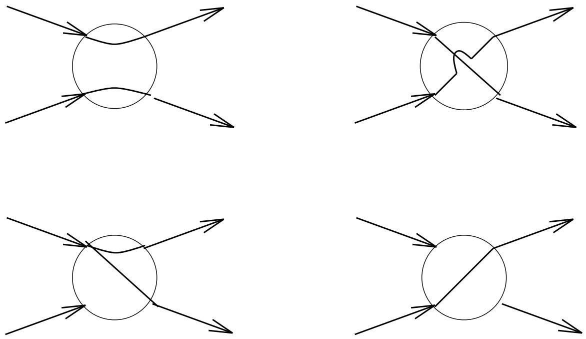

Figure 4-6: Some switching actions

A proof that the circuit is satisfiable should consist of an assignment of 0's and 1's to the inputs and the gates of the circuit that is consistent with a satisfying assignment (See Figure 4-9). The translation of this proof into a second coloring of the graph will consist of an assignment of 0's and 1's to the wires entering and leaving the nodes of the graph that is consistent with the assignment, the actions of the gates, and the switching actions. Since we are only supposed to color nodes of the graph, and not its wires, the proof will actually assign each node a four-tuple of symbols that indicate whether the edges attached to the node are 0, 1, or blank. (Figure 4-8 contains some valid second colors). We will choose our coloring rules so that the only legal colorings of the graph will be those that assign a 0 or 1 to each input, correctly propagate values along wires, correctly compute the value of every gate, and produce a 1 at the output gate. Figure 4-9 contains a picture of a proof that the circuit in Figure 4-5 is satisfiable. Figure 4-10 contains depictions of parts of second colorings that violate the coloring rules.

Now, we have fully described what constitutes a graph *coloring problem instance* and a graph *coloring problem solution*.

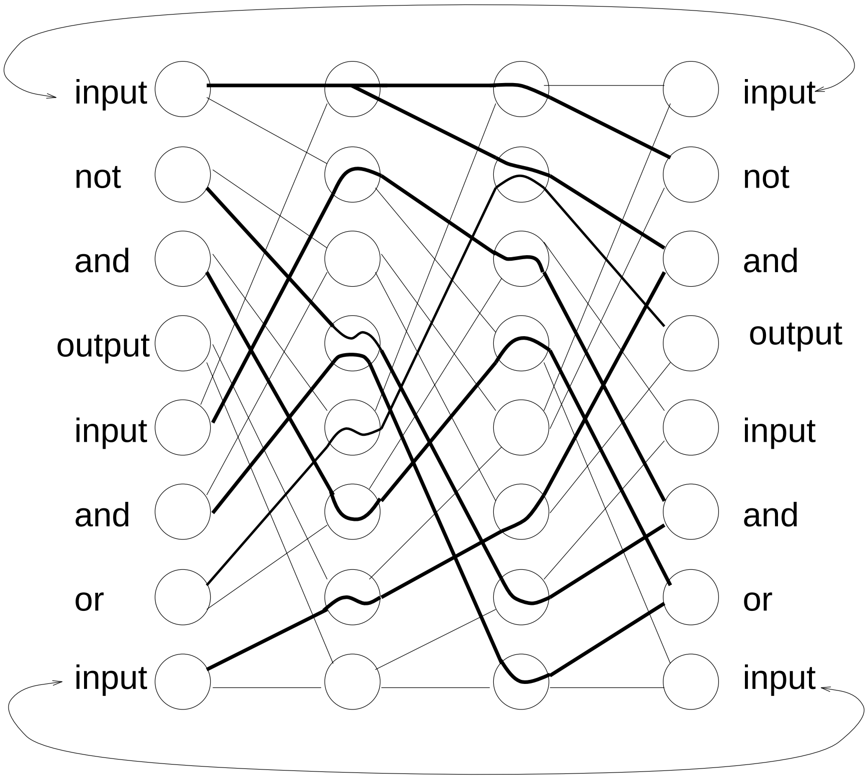

Figure 4-7: A drawing of the circuit on a wrapped de Bruijn graph. The left and right columns of the graph should be identified with one another, but are drawn twice for the convenience of the artist. Dark lines correspond to wires in the circuit.

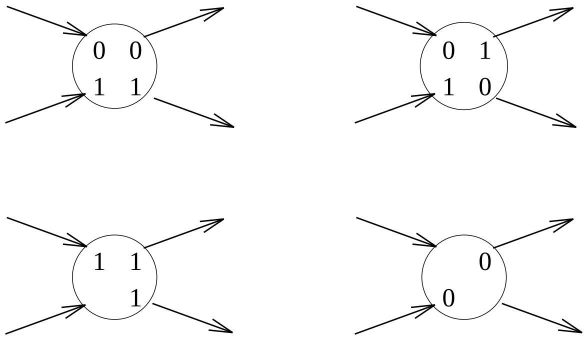

Figure 4-8: Some legal second colors for the switches in Figure 4-6.

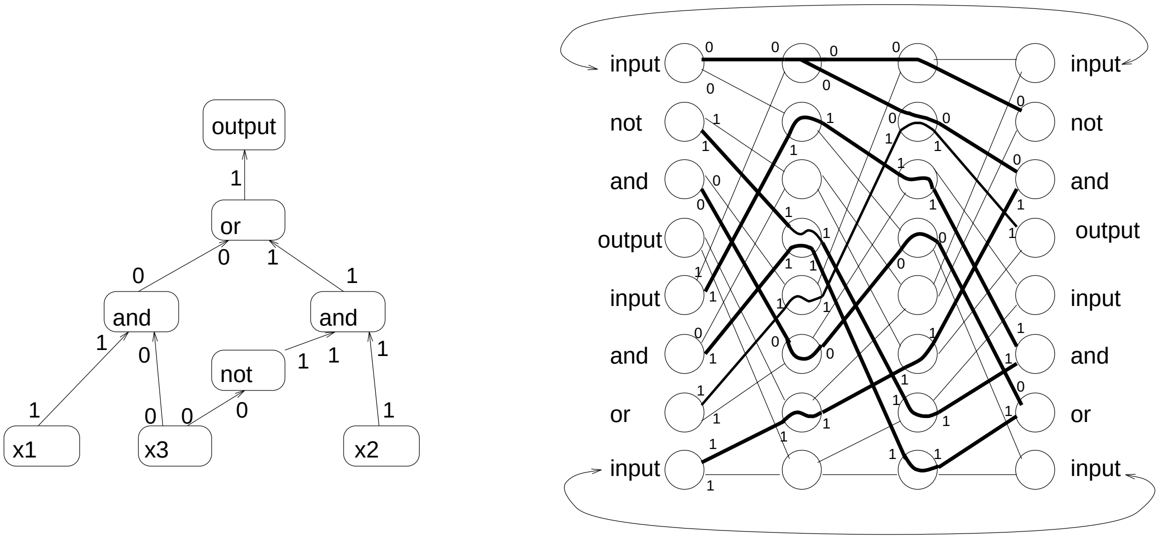

Figure 4-9: Two proofs that one circuit is satisfied.

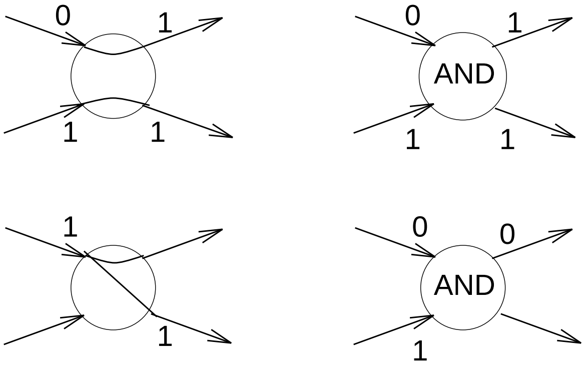

Figure 4-10: Some assignments of second colors that violate the coloring rules.

Remark 4.3.3 The length of this type of description of a circuit only differs from the length of a more conventional description by a constant factor. When one usually describes a circuit of m gates, one assigns a name to each gate and, for each gate, lists the operation that the gate performs and the names of the gates that are its inputs. This description should have size $\theta(m \log m)$. Since we describe a circuit of m gates by assigning one of a constant number of colors to each node of a graph of size $O(m \log m)$, these descriptions differ in size by at most a constant multiplicative factor.

4.3.2 Algebraically simple graphs

In this section, we will show that wrapped de Bruijn graphs have a very simple algebraic description. Actually, we obtain an algebraic description of graphs that we call *extended de Bruijn graphs*, which are the product of the line graph on $5n+1$ nodes and the de Bruijn graph. The identification of the first and last columns is provided by the holographic proof. We will use this description to translate our graph coloring problem into an algebraic problem to which we can apply the machinery we developed in Section 4.2. We begin by defining a graph on the elements of $GF(2^n)$.

Definition 4.3.4 A *Galois graph* G_n is a directed graph on 2^n nodes in which each node is identified with an element of $GF(2^n)$. Let α be a generator[7] of $GF(2^n)$. The node represented by $\gamma \in GF(2^n)$ has edges pointing to the nodes represented by

$$\alpha\gamma \qquad \text{and} \qquad \alpha\gamma + 1.$$

Lemma 4.3.5 The Galois graph G_n is isomorphic to the de Bruijn graph B_n.

Proof: We will begin by recalling a standard representation (the non-technical meaning of representation) of $GF(2^n)$. Let $p(\alpha)$ be an irreducible polynomial of degree n in $GF(2)[\alpha]$ (*i.e.* a polynomial with coefficients in $GF(2)$ that is irreducible over $GF(2)$).

$$GF(2^n) = GF(2)[\alpha]/p(\alpha).$$

[7] A generator is an element such that $\alpha^{2^n-1} = 1$ and $\alpha^k \neq 1$ for any $0 < k < 2^n - 1$. Every element of $GF(2^n)$ can be represented by a polynomial of degree less than n in α with coefficients in $\{0,1\}$.

That is, the elements of $GF(2^n)$ can be represented as polynomials in α of degree at most $n-1$ with coefficients in $GF(2)$. Addition of these polynomials is performed component-wise. To understand how multiplication behaves, let

$$p(\alpha) = \alpha^n + c_1\alpha^{n-1} + \cdots + c_{n-1}\alpha + c_n.$$

We multiply elements of $GF(2^n)$ by the standard multiplication algorithm for polynomials (remember to add component-wise). If we obtain a polynomial of degree higher than $n-1$, we apply the relation

$$\alpha^n = c_1\alpha^{n-1} + \cdots + c_{n-1}\alpha + c_n$$

until we obtain a polynomial of degree at most $n-1$.

We will represent the vertices of the de Bruijn graph on 2^n vertices by length n vectors of elements of $GF(2)$.

We define ϕ, and isomorphism from vertices of the de Bruijn graph to vertices of G_n, by

$$\phi(b_1, b_2, \ldots, b_n) = \alpha^{n-1}b_1 + \alpha^{n-2}(b_2 + c_1b_1) + \cdots + \left(b_n + \sum_{i=1}^{n-1} c_ib_{n-i}\right).$$

To see that this is an isomorphism, observe that in the de Bruijn graph, the edges leaving $(b_1, \ldots, b_n)$ go to vertices

$$(b_2, b_3, \ldots, b_n, b_1) \text{ and } (b_2, b_3, \ldots, b_n, b_1 \oplus 1);$$

whereas in G_n, one edge leaving

$$\alpha^{n-1}b_1 + \alpha^{n-2}(b_2 + c_1b_1) + \cdots + \left(b_n + \sum_{i=1}^{n-1} c_ib_{n-i}\right)$$

goes to

$$\alpha\left(\alpha^{n-1}b_1 + \alpha^{n-2}(b_2 + c_1b_1) + \cdots + \left(b_n + \sum_{i=1}^{n-1} c_ib_{n-i}\right)\right)$$

$$
\begin{aligned}
&= b_1\left(c_1\alpha^{n-1} + \cdots + c_{n-1}\alpha + c_n\right) + \alpha\left(\alpha^{n-2}(b_2 + c_1b_1) + \cdots + \left(b_n + \sum_{i=1}^{n-1} c_ib_{n-i}\right)\right) \\
&= \alpha^{n-1}b_2 + \alpha^{n-2}(b_3 + c_1b_2)\cdots + \alpha\left(b_n + \sum_{i=1}^{n-1} c_ib_{n-i}\right) + c_nb_1,
\end{aligned}
$$

and the other goes to

$$\alpha^{n-1}b_2 + \alpha^{n-2}(b_3 + c_1b_2)\cdots + \alpha\left(b_n + \sum_{i=1}^{n-1} c_ib_{n-i}\right) + c_nb_1 + 1,$$

which we can easily verify are

$$\phi(b_2, b_3, \ldots, b_n, b_1) \text{ and } \phi(b_2, b_3, \ldots, b_n, b_1 \oplus 1).$$

■

In our constructions of holographic proofs, we will actually want to identify the de Bruijn graph with a graph over $GF(2^{n/2}) \times GF(2^{n/2})$ rather than with a graph over $GF(2^n)$. This is easily accomplished:

Proposition 4.3.6 Let n be even and let α be a generator of $GF(2^{n/2})$. Then, the graph on $GF(2^{n/2}) \times GF(2^{n/2})$ in which the node represented by (σ, τ) has edges to the nodes represented by

$$(\tau, \alpha\sigma) \qquad \text{and} \qquad (\tau, \alpha\sigma + 1),$$

is isomorphic to the de Bruijn graph B_n.

Proof: By Lemma 4.3.5, we see that this graph is isomorphic to the graph on binary strings of length n in which the node $(b_1, \ldots, b_{n/2}, b_{n/2+1}, \ldots, b_n)$ has edges to the nodes

$$(b_{n/2+1}, \ldots, b_n, b_2, \ldots, b_{n/2}, b_1) \quad \text{and} \quad (b_{n/2+1}, \ldots, b_n, b_2, \ldots, b_{n/2}, b_1 \oplus 1).$$

We now shuffle the b_i's to see that this graph is isomorphic to the graph in which node $(b_1, b_{n/2+1}, b_2, b_{n/2+2}, \ldots, b_{n/2}, b_n)$ has edges to the nodes

$$(b_{n/2+1}, b_2, b_{n/2+2}, \ldots, b_{n/2}, b_n, b_1) \quad \text{and} \quad (b_{n/2+1}, b_2, b_{n/2+2}, \ldots, b_{n/2}, b_n, b_1 \oplus 1),$$

which is easily seen to be identical to the de Bruijn graph B_n. ■

We will conclude this section by presenting a simple algebraic description of the extended de Bruijn graph.

Proposition 4.3.7 Let n be even and let α be a generator of $\mathcal{F} = GF(2^{n/2})$. Let $\mathcal{E} = \{1, \alpha, \ldots, \alpha^{5n}\}$, and let $\mathcal{E}' = \{1, \alpha, \ldots, \alpha^{5n-1}\}$. Then, the extended de Bruijn graph on $(5n+1)2^n$ vertices is isomorphic to the graph on $\mathcal{F} \times \mathcal{F} \times \mathcal{E}$ in which each vertex in $(x, y, z) \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}'$ has edges to vertices

$$(y, \alpha x, \alpha z) \quad \text{and} \quad (y, \alpha x + 1, \alpha z).$$

For convenience, we will often denote a triple (x, y, z) by a vector, $\vec{x}$. We will then use $\rho_1(\vec{x})$ and $\rho_2(\vec{x})$ to denote the neighbors of $\vec{x}$. Formally, ρ_1 and ρ_2 are given by

$$\begin{aligned} \rho_1 : (x, y, z) &\mapsto (y, \alpha x, \alpha z) \\ \rho_2 : (x, y, z) &\mapsto (y, \alpha x + 1, \alpha z) \end{aligned}$$

4.3.3 Arithmetizing the graph coloring problem

In this section, we will use the algebraic description of the extended de Bruijn graphs given by Proposition 4.3.7 to construct an algebraic version of our graph coloring problem. For the purposes of this construction, we will define $\mathcal{F}$, $\mathcal{E}$, and $\mathcal{E}'$ as we did in Proposition 4.3.7.

Since we have identified the nodes of our graph with $\mathcal{F} \times \mathcal{F} \times \mathcal{E}$, we will view a coloring of the graph as a function over this domain. To this end, we will choose a set $\mathcal{C} \subset \mathcal{F}$ to represent the set of allowable colors. Thus, a coloring problem instance can be viewed as a function

$$T : \mathcal{F} \times \mathcal{F} \times \mathcal{E} \rightarrow \mathcal{C}.$$

So that it will jibe better with the machinery that we have developed, we will actually view the coloring problem instance as a degree $(|\mathcal{F}|, |\mathcal{F}|, |\mathcal{E}|)$ polynomial over $\mathcal{F}$. We will similarly view the coloring problem solution, $P(x, y, z)$, as a degree $(|\mathcal{F}|, |\mathcal{F}|, |\mathcal{E}|)$ polynomial. When we speak of a *presentation of a coloring problem solution*, we mean a presentation of this polynomial.

The local graph coloring rules will be described by a constant degree polynomial that takes a constant number of variables. Let $\vec{x} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}'$ be a vertex of the graph. Assuming that $T(\vec{x})$, $T(\rho_1(\vec{x}))$, $T(\rho_2(\vec{x}))$, $P(\vec{x})$, $P(\rho_1(\vec{x}))$, and $P(\rho_2(\vec{x}))$ are all in $\mathcal{C}$, we can form a constant degree polynomial χ which has the property that

$$\chi\left(T(\vec{x}), T(\rho_1(\vec{x})), T(\rho_2(\vec{x})), P(\vec{x}), P(\rho_1(\vec{x})), P(\rho_2(\vec{x}))\right) = 0$$

if and only if the colors assigned by T and P to $\vec{x}$, $\rho_1(\vec{x})$, and $\rho_2(\vec{x})$ satisfy all the local coloring rules. The reason that we can assume that χ has constant degree is that its value is constrained only at the points of $\mathcal{C}^6$, which is a finite set. Since the coloring rules are the same throughout the graph, χ does not depend on $\vec{x}$.

In order to check that the polynomials T and P actually do map each node of the graph to an element of $\mathcal{C}$, rather than an arbitrary element of $\mathcal{F}$, we will form a constant degree univariate polynomial $\psi(\gamma)$ which will be zero if and only if $\gamma \in \mathcal{C}$.[8] Before we check that the coloring rules described by χ are satisfied, we will first check that $\psi(P(\vec{x}))$ and $\psi(T(\vec{x}))$ evaluate to zero for each $\vec{x} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}$.

To check that the colors assigned to the last column of the extended de Bruijn graph are the same as the colors assigned to the first column, we need merely check that for all $(x, y) \in \mathcal{F} \times \mathcal{F}$,

$$T(x, y, 1) - T(x, y, \alpha^{5n}) = 0 \qquad \text{and} \qquad P(x, y, 1) - P(x, y, \alpha^{5n}) = 0.$$

We can now re-state the graph coloring problem as a problem concerning the existence of certain polynomials: Given a degree $(|\mathcal{F}|, |\mathcal{F}|, |\mathcal{E}|)$ polynomial T over $\mathcal{F}$, we say that another degree $(|\mathcal{F}|, |\mathcal{F}|, |\mathcal{E}|)$ polynomial P *solves* T if

$$\begin{array}{lll} (1) & \psi(P(\vec{x})) = 0 \text{ and } \psi(T(\vec{x})) = 0, & \forall \vec{x} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E} \\ (2) & \chi\left(T(\vec{x}), T(\rho_1(\vec{x})), T(\rho_2(\vec{x})), P(\vec{x}), P(\rho_1(\vec{x})), P(\rho_2(\vec{x}))\right) = 0, & \forall \vec{x} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}' \\ (3) & T(x, y, 1) - T(x, y, \alpha^{5n}) = 0 \text{ and } P(x, y, 1) - P(x, y, \alpha^{5n}) = 0, & \forall (x, y) \in \mathcal{F} \times \mathcal{F} \end{array}$$

[8] We can choose $\psi(\gamma) = \prod_{c \in \mathcal{C}}(\gamma - c)$.

We can view the polynomial T as a coloring problem instance and the polynomial P as a coloring problem solution. We won't verify that all these conditions are satisfied by examining the values of T and P at each point of $\mathcal{F} \times \mathcal{F} \times \mathcal{E}$ individually. Instead, we will provide presentations of the polynomials $T(x, y, z)$ and $P(x, y, z)$, as well as some additional information, so that the proof checker will only need to examine only a constant number of segments of each presentation. The details of this process are explained in the next section.

4.3.4 A Holographic Proof

We will want to provide presentations of $P(x, y, z)$ and $T(x, y, z)$ that are verifiable (see Section 4.2.5). To do this, we will choose domains $\mathcal{H}$ and $\mathcal{J}$ that contain $\mathcal{F}$ and $\mathcal{E}$ respectively, and insist that P and T be presented over $\mathcal{H} \times \mathcal{H} \times \mathcal{J}$. The conditions of Lemma 4.2.24 require that $\mathcal{H}$ and $\mathcal{J}$ have sizes that are larger than $\mathcal{F}$ and $\mathcal{E}$ by a constant factor. Other constraints on our choice of $\mathcal{H}$ and $\mathcal{J}$ will appear later in this section. For now, let us fix a field $\mathcal{G}$ that contains the field $\mathcal{F}$, and insist that $\mathcal{H}$ and $\mathcal{J}$ be subsets of $\mathcal{G}$. Note that it is possible to find such a field $\mathcal{G}$ whose size is the square of the size of $\mathcal{F}$.[9] We now describe what the proof checker should do to verify that P solves T, and what needs to be provided with P to enable to proof checker to do so. To anthropomorphize the situation, we imagine that a proof provider wants to supply the proof checker with enough information to check that P solves T.

The techniques that we use in this section are derived from [Sud92].

Remark 4.3.8 Various unspecified constants will appear in this discussion. It should be clear that sufficiently extreme choices for these constants will suffice to prove the lemma that concludes the section.

STEP 1: When provided with presentations of T and P, the proof checker should check that these presentations are ϵ-good, for some small constant ϵ.

Remark 4.3.9 Every time the proof checker is provided with a presentation by the proof provider, it will check that the presentation is ϵ-good for some very small ϵ. If the presentation

[9] It is an elementary result from the theory of finite fields that $GF(2^{n/2})$ is a subfield of $GF(2^n)$.

passes this test, then the proof checker will thereafter assume that the presentation actually does represent some polynomial and that, whenever it queries a segment of the presentation, it receives a segment of the presentation of that polynomial. We will choose ϵ to be sufficiently small so that if the proof checker were mistaken in this assumption, this mistake would have been detected with high probability during the checking.

We will now describe how the proof checker should verify that P and T satisfy condition (1).

Remark 4.3.10 Actually, the proof checker can only become confident that the polynomials to which the presentations of P and T are close satisfy relation (1). It cannot know anything about any particular piece of these presentations that it does not read. This is an important thought to keep in mind as one reads this section.

The proof checker will insist that the proof provider provide a presentation of the polynomials $\psi(P(x,y,z))$ and $\psi(T(x,y,z))$ on the domain $\mathcal{H} \times \mathcal{H} \times \mathcal{J}$. We will choose $\mathcal{H}$ and $\mathcal{J}$ sufficiently large so that these presentations are checkable and verifiable. Since ψ is a constant degree polynomial, the degrees of $\psi(P(x,y,z))$ and $\psi(T(x,y,z))$ will be only be larger than the degrees of $P(x,y,z)$ and $T(x,y,z)$ by a constant factor. Because the proof checker cannot be sure that the presentations that it is given actually do represent $\psi(P)$ and $\psi(T)$, we will refer to the presentations that the proof provider provides as $\widehat{\psi(P)}$ and $\widehat{\psi(T)}$.

STEP 2: The proof checker should check that the presentations $\widehat{\psi(T)}$ and $\widehat{\psi(P)}$ ϵ-good, for some small constant ϵ.

Now that the proof checker has checked that the presentations of P, T, $\widehat{\psi(P)}$, and $\widehat{\psi(T)}$ are good, it will assume that they are close to presentations of polynomials of the appropriate degrees, and it will want to check that these polynomials actually are $\psi(P)$ and $\psi(T)$. It will do this by choosing random points $\vec{x}$, and verifying that $\psi(P(\vec{x})) = \widehat{\psi(P)}(\vec{x})$. It will then do the same for T.

STEP 3: The proof checker should choose a constant number of points $\{\vec{x}_1, \ldots, \vec{x}_c\} \subset \mathcal{H} \times \mathcal{H} \times \mathcal{J}$. For each point $\vec{x}_i$, the proof checker should read $T(x_i)$ and $P(x_i)$, compute

$\psi(T(\vec{x}_i))$ and $\psi(P(\vec{x}_i))$, and then check that these values agree with $\widehat{\psi(P)}(\vec{x}_i)$ and $\widehat{\psi(T)}(\vec{x}_i)$.

From Lemma 4.2.5, we know that if $\widehat{\psi(P)}$ and $\widehat{\psi(T)}$ were presentations of polynomials other than $\psi(P)$ and $\psi(T)$, then they would fail Step 3 with high probability. Thus, once the presentations have passed Step 3, the proof checker will be safe in assuming that whenever it reads a segment from $\widehat{\psi(P)}$ or $\widehat{\psi(T)}$, it actually reads a segment of the presentation of $\psi(P)$ of $\psi(T)$.

We will now describe how the proof provider will convince the proof checker that $\psi(P(\vec{x})) = 0$ for all $\vec{x} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E}$. The proof for $\psi(T)$ will be similar, so we will just discuss the proof for $\psi(P)$.

The proof provider will provide presentations of polynomials that we will call Ψ', Ψ'', and Ψ''', where these polynomials are defined to be:

$$\begin{aligned}
\Psi'(x,y,z) &= \sum_{i=1}^{2^{r/2}} \psi(P(f_i,y,z))x^{i-1} \qquad (*) \\
\Psi''(x,y,z) &= \sum_{j=1}^{2^{r/2}} \Psi'(x,f_j,z)y^{j-1} \\
\Psi'''(x,y,z) &= \sum_{l=0}^{5r} \Psi''(x,y,\alpha^l)z^l,
\end{aligned}$$

where $\{f_1,\ldots,f_{2^{n/2}}\}$ are the elements of $\mathcal{F}$. The purpose of these polynomials is best understood by observing that

$$\Psi'''(x,y,z) = \sum_{i=1}^{2^{r/2}} \sum_{j=1}^{2^{r/2}} \sum_{l=0}^{5r} \psi(P(f_i,f_j,\alpha^l))x^{i-1}y^{j-1}z^l.$$

The polynomial Ψ''' is the zero polynomial if and only if $\psi(P)$ is zero on all of $\mathcal{F} \times \mathcal{F} \times \mathcal{E}$. Moreover, we can efficiently check that each of Ψ', Ψ'', and Ψ''' are formed correctly and that Ψ''' is the zero polynomial. We will call $\widehat{\Psi'}$, $\widehat{\Psi''}$, and $\widehat{\Psi'''}$ the presentations that the proof provider provides which it claims are presentations of Ψ', Ψ'', and Ψ'''.

STEP 4: The proof checker should check that the presentations $\widehat{\Psi'}$, $\widehat{\Psi''}$, and $\widehat{\Psi'''}$ are ϵ-good, for some small constant ϵ. The proof checker should do the same for the analogous polynomials corresponding to $\psi(T)$.

STEP 5: The proof checker should choose a constant number of random points

$$\{(y_1, z_1), \ldots, (y_c, z_c)\} \in \mathcal{H} \times \mathcal{J}.$$

For each point (y_i, z_i), the proof checker should check that the univariate polynomial in x through (y_i, z_i) in the presentation of Ψ' and the corresponding polynomial in the presentation of $\psi(P)$ have the relation indicated by (*). The proof checker should then perform the analogous operations to verify that Ψ'' and Ψ''' have been properly formed.

The proof checker can efficiently verify that the two univariate polynomials through (y_i, z_i) have the relation indicated in (*) by performing a Finite Fourier transform. This test is sufficient to convince the proof checker that Ψ' has been correctly formed because, after Step 4, the proof checker is confident that the presentation that purports to be Ψ' is a presentation of the appropriate degree and Lemma 4.2.5 implies that the test in Step 5 would fail with high probability if Ψ' were not the correct polynomial. The proof checker is similarly convinced that $\widehat{\Psi''}$ and $\widehat{\Psi'''}$ are presentations of the polynomials Ψ'' and Ψ''' and that they satisfy the desired relations with $\psi(P)$.

STEP 6: The proof checker should choose a constant number of points

$$\{\vec{x}_1, \ldots, \vec{x}_c\} \in \mathcal{F} \times \mathcal{F} \times \mathcal{E},$$

and verify that $\Psi'''(\vec{x}_i) = 0$, for each i.

After performing Step 6, the proof checker will be confident that $\Psi''' \equiv 0$ (this is another application of Lemma 4.2.5). If the proof has passed all these steps, then the proof checker can be confident that the presentations of P and T satisfy condition (1).

We will now explain how the proof checker will verify that P and T satisfy condition (2). This procedure will be very similar to the procedure for condition (1), so we will only describe in detail those places in which it differs.

The proof checker will need access to presentations of $T(\rho_1(\vec{x}))$, $T(\rho_2(\vec{x}))$, $P(\rho_1(\vec{x}))$, and $P(\rho_2(\vec{x}))$. We will choose $\mathcal{H}$ and $\mathcal{J}$ so that these presentations are contained as sub-presentations of the presentations of T and P. Let $\mathcal{I}$ be a set so that $\mathcal{F} \subset \mathcal{I}$, $\mathcal{I} = \mathcal{I} + 1$, and $\mathcal{I}$ is some constant times larger than $\mathcal{F}$. We will let $\mathcal{H} = \mathcal{I} \cup \alpha\mathcal{I}$. We similarly choose $\mathcal{K}$ to be a set which contains $\mathcal{E}$ and which is larger by a constant factor. We then set $\mathcal{J} = \mathcal{K} \cup \alpha\mathcal{K}$. We now observe that the presentations of T and P on $\mathcal{H} \times \mathcal{H} \times \mathcal{J}$ contain sub-presentations of $T(\vec{x})$, $T(\rho_1(\vec{x}))$, $T(\rho_2(\vec{x}))$, $P(\vec{x})$ $P(\rho_1(\vec{x}))$, and $P(\rho_2(\vec{x}))$ on $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$. Proposition 4.2.23 tells us that once we have verified that the presentations of P and T over $\mathcal{H} \times \mathcal{H} \times \mathcal{J}$ are ϵ-good, we know that these sub-presentations over $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$ are 8ϵ-good. Since it verified that the presentations of P and T were good in Step 1, the proof checker will assume that these other presentations are good as well, and that whenever it queries one of these presentations at a point, it receives the value of the polynomial to which the presentation is close.

The proof provider should provide a presentation of

$$\chi\left(T(\vec{x}), T(\rho_1(\vec{x})), T(\rho_2(\vec{x})), P(\vec{x}), P(\rho_1(\vec{x})), P(\rho_2(\vec{x}))\right)$$

on $\mathcal{I} \times \mathcal{I} \times \mathcal{K}$. We will call this presentation $\widehat{\chi}$.

STEP 7: The proof checker should check that the presentation of $\widehat{\chi}$ is ϵ-good for some small ϵ. This is analogous to Step 2.

Since χ, ρ_1 and ρ_2 are constant degree polynomials, the domains $\mathcal{I}$ and $\mathcal{K}$ only need to be larger than $\mathcal{F}$ and $\mathcal{E}$ by a constant multiplicative factor in order to make this step work.

Now that the proof checker knows that $\widehat{\chi}$ is close to the presentation of some polynomial, it needs to verify that this polynomial has been built correctly from the presentations of T and P.

STEP 8: This step is analogous to Step 3. The proof checker should choose a constant number of points $\{\vec{x}_1, \ldots, \vec{x}_c\} \subset \mathcal{I} \times \mathcal{I} \times \mathcal{K}$. For each point $\vec{x}_i$, the proof checker should read $T(\vec{x}_i)$, $T(\rho_1(\vec{x}_i))$, $T(\rho_2(\vec{x}_i))$, $P(\vec{x}_i)$ $P(\rho_1(\vec{x}_i))$, and $P(\rho_2(\vec{x}_i))$, and and then check that

$$\widehat{\chi}(\vec{x}_i) = \chi\left(T(\vec{x}_i), T(\rho_1(\vec{x}_i)), T(\rho_2(\vec{x}_i)), P(\vec{x}_i), P(\rho_1(\vec{x}_i)), P(\rho_2(\vec{x}_i))\right)$$

for each i.

From Lemma 4.2.5, we know that if $\widehat{\chi}$ is not close to the correct polynomial, then this step will fail with high probability. Thus, once the presentation has passed Step 8, the proof checker will be safe in assuming that whenever it reads a segment from $\widehat{\chi}$, it actually receives a segment of the presentation of χ.

Now, the proof checker must check that χ is zero over $\mathcal{F} \times \mathcal{F} \times \mathcal{E}'$. This is done in exactly the same was it was done for ψ.

STEPS 9, 10, and 11: Analogous to Steps 4, 5, and 6.

It only remains to check that condition (3) is satisfied. We will only explain how the proof checker does this for P as the procedure for T is identical.

STEPS 12: Check that the sub-presentations of P on $\mathcal{H} \times \mathcal{H} \times 1$ and $\mathcal{H} \times \mathcal{H} \times \alpha^{5n}$ are ϵ-good for some small ϵ. Check these on the same points and make sure that they agree on these points.

STEPS 13: Check that the bivariate polynomials which the presentations of P on $\mathcal{H}\times\mathcal{H}\times 1$ and $\mathcal{H} \times \mathcal{H} \times \alpha^{5n}$ are close to are the restrictions of the trivariate polynomial to which the presentation of P is close using Step 3 of the trivariate verification algorithm.

Note that if the proof checker does not perform Step 13, then it would be possible for the sub-presentations on $\mathcal{H} \times \mathcal{H} \times 1$ and $\mathcal{H} \times \mathcal{H} \times \alpha^{5n}$ to be identical but have nothing to do with the rest of the presentation of P.

We have now completed our description of our first holographic proof. Observe that the proof consists only of a constant number of presentations of polynomials on domains larger than $\mathcal{F} \times \mathcal{F} \times \mathcal{E}$ by a constant factor. We have proved:

Lemma 4.3.11 There exists a probabilistic algorithm V that expects as input a presentation of a coloring problem instance, a presentation of a coloring problem solution, and a constant number of additional presentations such that:

- we refer to the the presentation of the coloring problem solution and the additional presentations as a holographic proof;
- the total size of the presentations in the holographic proof is at most $n \log^{O(1)} n$, where n is size of the coloring problem instance;

- V only reads a constant number of segments, each of size at most $\sqrt{n}\log^{O(1)} n$, from each presentation;
- if the coloring problem instance is solvable, then there is a holographic proof that will cause V to accept with probability 1;
- if V accepts with probability greater than $1/2$, then the inputs of V are close to a presentation of a coloring problem instance and a proof that that instance is solvable.

In the next section, we will see how to recursively apply this proof system to itself.

4.4 Recursion

In this section, we will see how to recursively apply the holographic proofs constructed in Section 4.3.

We begin by examining the operations that are performed by the checker of the holographic proofs constructed in Section 4.3.4 (*i.e.* the operations in Steps 1–13). We observe that only four types of operations are performed by the checker:

- checking whether a given polynomial is zero at a certain point,
- checking whether a given polynomial assumes a given value at a certain point,
- checking whether some constant degree polynomial evaluated at a given point takes a given value, and
- in Step 5, checking whether a given polynomial assumes a given set of values at a set of points, where the size of that set is smaller than the degree of the polynomial.

All of these operations can be performed by circuits of size $n \log^{O(1)} n$. The main idea behind our recursion will be to provide holographic proofs that each of these operations can be performed correctly so that the checker doesn't actually have to perform the operations. Thus, instead of reading a constant number of segments of size roughly $O(\sqrt{n})$ to check a holographic proof, the checker will be able to read a constant number of segments of size $O(\sqrt[4]{n})$ for each of the original segments. After k levels of recursion the checker will read $2^{O(k)}$ segments of size roughly $O(\sqrt[2^k]{n})$. We can then cap this recursion with Theorem 4.0.1 to obtain holographic proofs of size $O(n^{1+\epsilon})$ checkable by a constant number of queries, for any $\epsilon > 0$.

The two main tools that we will use to implement this idea are "encoded inputs" and the Fast Fourier Transform.

4.4.1 Encoded inputs

When Babai, Fortnow, Levin, and Szegedy [BFLS91] introduced transparent proofs, they presented the theorem candidate in an error-correcting code so that the proof checker would only have to read a constant number its bits. This idea was vital to the recursions in the work of Arora and Safra [AS92b] and Arora, Lund, Motwani, Sudan, and Szegedy [ALM$^+$92], and we will make use of it as well.

In Section 4.3, we saw how to create a holographic proof that a circuit has a satisfying assignment. A weakness of this construction is that the satisfying assignment appears within the holographic proof. For our recursion, we will want to construct many holographic proofs concerning one piece of data. Thus, we will need some way of checking that part of a satisfying assignment in a holographic proof is the same as a piece of external data. We want to do this without reading more than a constant number of bits of the external data.

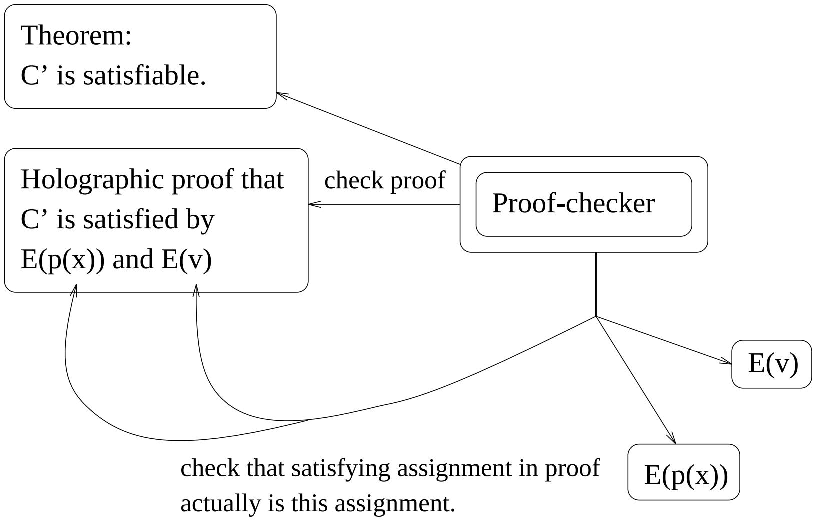

Figure 4-11: The proof checker needs to be sure that the proof shows that C' is satisfied by $E(p(x))$ and $E(v)$, which are written elsewhere.

For example, we might want to provide a holographic proof that a polynomial $p(x)$ takes the value v at x_0, but we don't want to force the proof checker to read the entire description of $p(x)$ or v. To do this, we first choose an asymptotically good error-correcting code, which we will call E for the rest of the chapter.[10] Instead of writing the description of $p(x)$ and v in the normal way, we insist that they be presented as $E(p(x))$ and $E(v)$. Now, consider a circuit C that takes a polynomial and a value as input and accepts if that polynomial has that value when evaluated at x_0. We modify C to produce a circuit C' that expects its

[10] Actually, it is unnecessary that E have constant rate, but we may as well assume that it does.

inputs to be encoded in the code E. C' will accept if and only if its inputs are codewords of E that encode a polynomial and a value that would cause C to accept. Our checker will ask for a holographic proof that C' accepts on inputs $E(p(x))$ and $E(v)$. When the checker checks that the holographic proof is correct, it becomes convinced that there is a satisfying assignment of C', but it has not yet learned anything about that assignment. To become convinced that the satisfying assignment in the holographic proof actually is the same as its external data (which it presumes to be $E(p(x))$ and $E(v)$), the checker will read a few randomly chosen bits of its external data and check that these agree with the corresponding inputs represented in the holographic proof (see Figure 4-11). The checker can check the values of inputs represented in the holographic proof because the presentations in the proof are verifiable (as discussed in Section 4.1); so, the checker can verify the value of any gate in the circuit represented in the holographic proof.

If we choose E to be a code with constant relative minimum distance then, after successfully comparing a constant number of bits, the checker will be convinced that the external data is close to a unique codeword of E and that that codeword is the same as the one in the satisfying assignment in the holographic proof. This is because, after checking the holographic proof, the checker is confident that its satisfying assignment contains codewords of E, and all codewords of E are far apart so a piece of external data can agree with high probability with only one codeword of E. If we choose E to be a code in which it is easy to verify whether or not a word is in the code, then the circuit C' will have almost the same size as the circuit C. The codes that we constructed in Chapters 2 and 3 can be decoded in linear time, so these are well-suited to our purposes. If we use one of these codes, then the size of C' will exceed the size of C by at most a constant times the size of its inputs. We assume that E is one of these codes in the rest of this chapter.

4.4.2 Using the Fast Fourier Transform

One's first inclination for how to recursively apply our construction of holographic proofs—encode every segment of every object in an error-correcting code and provide holographic proofs for each operation that the checker needs to perform—creates proofs that are a little too big. Consider what happens if we try this for the bivariate code that we presented in

Section 4.2. There are $\theta(\sqrt{n})$ univariate polynomials of degree $\theta(\sqrt{n})$ in a presentation containing $\theta(n)$ bits of data. The checker might want to evaluate any of these polynomials at any of $\theta(\sqrt{n})$ points. The holographic proof that a polynomial of degree $\theta(\sqrt{n})$ takes a certain value at a certain point will have size at least $\theta(\sqrt{n})$. If we were to provide a separate proof for each of the $\theta(\sqrt{n})$ polynomials at each of $\theta(\sqrt{n})$ points, then these proofs would have total size at least $\theta\left(n^{3/2}\right)$.

To prevent this explosion, we will combine many of these proofs into one. For each polynomial, we will create just one proof that provides its values when it is evaluated at all of the points we are concerned with. We replace each presentation of a degree (d_x, d_y, d_z) polynomial p over a domain $X \times Y \times Z$ with (see Figure 4-12)

- for each point in the domain, an encoding of the value of p at that point,
- for each univariate polynomial in the presentation, and encoding of the description of that polynomial, and
- for each univariate polynomial in the presentation, a holographic proof that presents its value for each point it intersects in the domain; this proof should be verifiable so that it can be used as a proof that any one of the encoded polynomials takes one of the encoded values at a point. The proof checker will need to provide a coloring problem instance that describes this relation.

For each of the four types of operations that the checker needs to perform (listed at the beginning of Section 4.4) we present a holographic proof that this operation has been performed correctly.

Aho, Hopcroft and Ullman [AHU74, Chapter 8.5] show that a polynomial can be efficiently evaluated at any reasonable set of points in a field using a few discrete fourier transforms. Using the efficient implementation of the discrete fourier transform over a finite field presented by Preparata and Sarwate [PS77], the univariate polynomials can be evaluated at each point in their domain by circuits of size $n \log^{O(1)} n$. Thus, if we use Lemma 4.3.11 to construct the holographic proofs, the total size of these objects is larger than the size of the presentation by at most a poly-logarithmic factor. Similarly, if we use

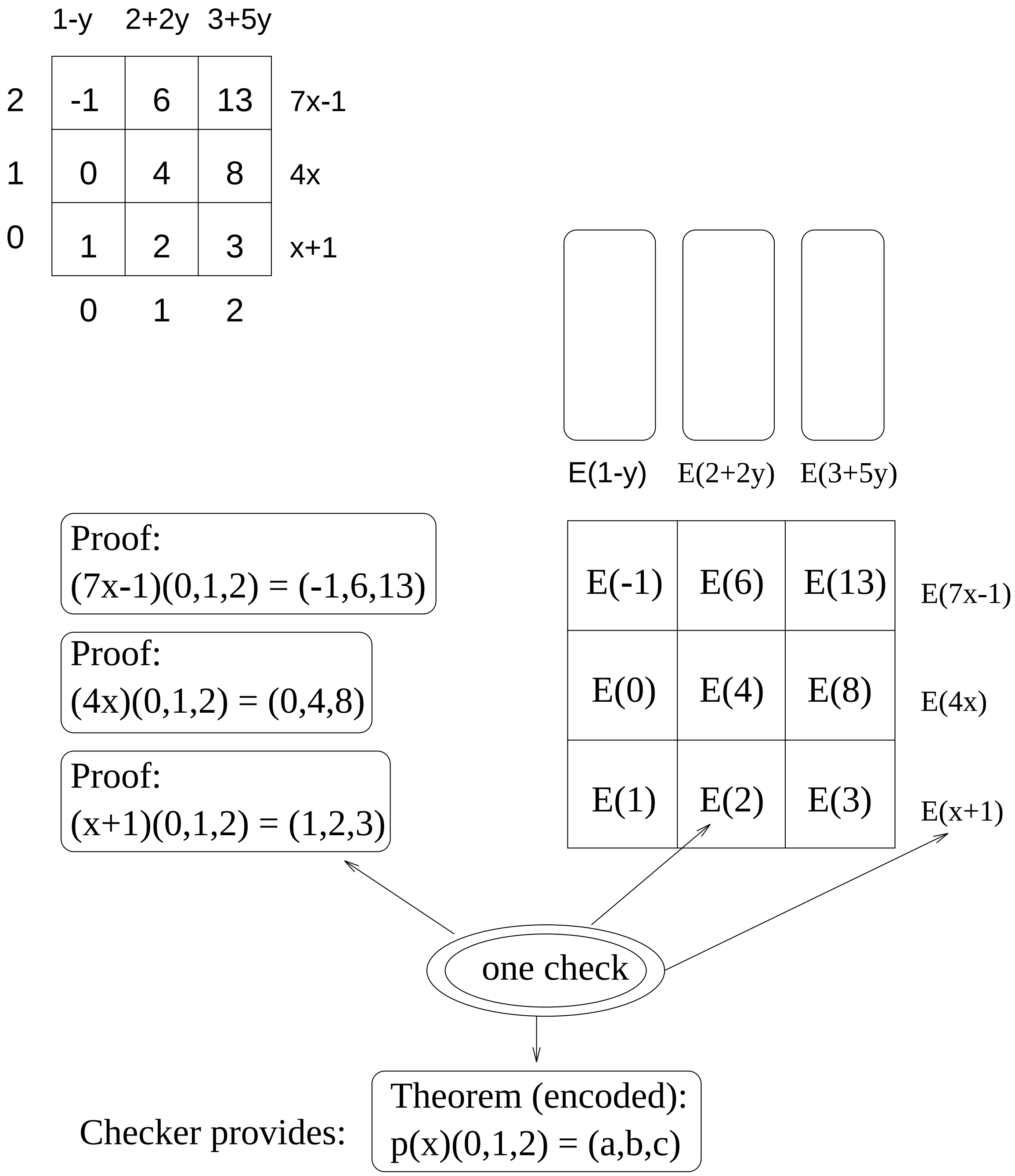

Figure 4-12: A presentation is replaced for recursion.

Theorem 4.0.1 to construct the holographic proofs, then the total size of these objects is at most a polynomial in the size of the presentation.

We can now prove the first variation of our main holographic proof theorem.

Theorem 4.4.1 [Efficient holographic proofs, variation 1] For all polynomial-time constructible functions $q(n) = O(\log\log n)$, there exists a polynomial time algorithm V (called a proof checker) that expects as input an encoding of a coloring problem instance T and a holographic proof Π such that

- $|\Pi| = |T|^{1+2^{O(-q(n))}} (\log |T|)^{O(q(n))}$;
- V reads only $2^{O(q(n))}$ bits of T and Π;
- if the coloring problem instance T is solvable, then there is a holographic proof Π that will cause V to accept with probability 1; moreover, Π can be computed from a coloring problem solution of T in time $|\Pi| \log^{O(1)} |\Pi|$;
- if V accepts (T, Π) with probability greater than $1/2$, then T is close to an encoding of a coloring problem instance and Π contains a presentation that is close to a solution of that instance.

Proof: We first apply Lemma 4.3.11 to itself $q(n)$ times. To perform the recursion, we replace each presentation by encodings of its objects and a constant number of additional presentations as described in Sections 4.4.1 and 4.4.2. Note that the checker must provide presentations of coloring problem instances that say that the operations in the recursion are being performed correctly, while the proof must contain presentations of coloring problem solutions. These presentations are verifiable so that the checker can make sure that they actually refer to the encoded objects.

At this point, we have a proof Π that has size $|T| (\log |T|)^{O(q(n))}$ which can be checked by examining $2^{O(q(n))}$ of its segments, each of which has size at most $|T|^{2^{-q(n)}} (\log |T|)^{O(q(n))}$. We now cap the recursion by applying Theorem 4.0.1 to construct the last set of holographic proofs. This will result in a polynomial blow-up in the size of each presentation. ■

4.5 Efficient proof checkers

In this section, we explain how to make the checkers of the proofs presented in section 4.4 run in poly-logarithmic time. We begin by examining the computations performed by the checker of Theorem 4.4.1. Most of the checker's computation is devoted to constructing the coloring problem instances that are needed in the recursion. Fortunately, these coloring problem instances do not depend on the actual coloring problem being proved solvable. They are only statements like "there is a polynomial of degree d that evaluates to some value at x_0." Thus, all of these presentations of coloring problem instances can be computed once and used in all proof checking to be done thereafter. Assuming that the proof checker has access to these presentations, it need only do poly-logarithmic work: the rest of its operations are choosing which parts of the proof to look at and checking proofs constructed by Theorem 4.0.1. Thus, we have proved the second variation of our holographic proof theorem:

Theorem 4.5.1 [Efficient holographic proofs, variation 2] For all polynomial-time constructible functions $q(n) = O(\log\log n)$, there exists a table A and a probabilistic algorithm V (called a proof checker) that expects as input an encoding of a coloring problem instance T and a holographic proof Π such that

- $|\Pi| = |T|^{1+2^{O(-q(n))}} (\log|T|)^{O(q(n))}$;
- V reads only $2^{O(q(n))}$ bits of T, Π, and A;
- if the coloring problem instance T is solvable, then there is a holographic proof Π that will cause V to accept with probability 1; moreover, Π can be computed from a coloring problem solution of T in time $|\Pi| \log^{O(1)} |\Pi|$;
- if V accepts (T, Π) with probability greater than $1/2$, then T is close to an encoding of a coloring problem instance and Π contains a presentation that is close to a solution of that instance;
- V runs in time $\log^{O(1)} |T|$.

For all practical purposes, our checker now runs in poly-logarithmic time. The table of encodings of coloring problem instances can be written once and accessed forever. However, the need for this table can be eliminated if we use a coloring problem developed in [BFLS91].

4.6 The coloring problem of Babai, Fortnow, Levin, and Szegedy

Babai, Fortnow, Levin, and Szegedy set out to provide holographic versions of any proof of any mathematical statement. To achieve this end, they rely on the Kolomorgorov-Uspenskiĭ thesis which implies that any proof can be efficiently expressed by a computation of a certain type of non-deterministic pointer machine [KU58]. They then observe that the computations of these pointer machines can be efficiently witnessed by restricted computations on a RAM. They translate these RAM computations into a coloring problem on a graph that is similar to the coloring problem we used in Section 4.3.1.[11] The advantage of the coloring problem in [BFLS91] is that the problem instance does not need to be as large as the problem solution.

In the coloring problem of [BFLS91], each node receives only one color. The problem instance is described by a coloring of an initial set of the nodes in the graph. The problem instance is solvable if there exists a coloring of the rest of the nodes in the graph that satisfies all the coloring rules. The type of problem instance that they have in mind is an encoding, by a good error-correcting code, of a proposed theorem. They demonstrate the existance of a set of local coloring rules such that it will be possible to complete the coloring of the graph in agreement with the rules only if the problem instance decodes to a statement of a provable theorem. Moreover, given a proof of the theorem, one can easily construct a coloring of the rest of the graph that satisfies the coloring rules. Actually, the techniques developed in [BFLS91] are even more general—they show that for any non-deterministic RAM algorithm, there is a finite set of coloring rules under which the rest of the nodes of the graph can be legally colored only if there is a computation of the algorithm that accepts on the input indicated by the initial coloring. The application to checking proofs is a special case in which the RAM algorithm checks proofs.

[11] Our coloring problem was inspired by theirs. It is weaker, but we hope it is easier to understand.

The graphs on which their coloring problem is defined are very similar to the graphs we used in Section 4.3.1: we can assume that their graphs are finite products of families of de Bruijn graphs and line graphs. Their coloring rules can be assumed to be the natural generalizations of coloring rules such as conditions (1), (2), and (3) of Section 4.3.1. Thus, instances of their graph coloring problem can be checked using algebraic techniques similar to those used in Sections 4.2, and 4.3.1.

The advantage of using the coloring problem of [BFLS91] in our recursion is that it eliminates the need for the checker to write coloring problem instances to check the recursion. For each type of relation that the checker needs to check, there is one uniform RAM algorithm that can perform the check. Thus, the checker need merely know the finite set of coloring rules associated with checking the computations associated with each of these algorithms, and verify that they are satisfied.

Thus, by combining the algebraic machinery developed in Section 4.2 and the recursion developed in Section 4.4 with the coloring problem of [BFLS91], we can prove the final variation of our holographic proof theorem:

Theorem 4.6.1 [Efficient holographic proofs, variation 3] For all polynomial-time constructible functions $q(n) = O(\log\log n)$, there exists a probabilistic algorithm V (called a proof checker) that expects as input an encoding of a theorem T, which we call a *theorem candidate*, and and a holographic proof Π such that

- if T encodes a theorem that has a proof P, then there is a holographic proof Π that will cause V to accept with probability 1; moreover, Π can be computed from P in time $|\Pi| \log^{O(1)} |\Pi|$;
- $|\Pi| = |P|^{1+2^{O(-q(n))}} (\log |P|)^{O(q(n))}$;
- V reads only $2^{O(q(n))}$ bits of T and Π;
- if V accepts (T, Π) with probability greater than $1/2$, then T is close to a unique encoding of a true theorem and Π constitutes a proof of that theorem;
- V runs in time $\log^{O(1)} |\Pi|$.

For convenience, we state the following corollary:

Corollary 4.6.2 For all $\epsilon > 0$, there exists a probabilistic polynomial-time turing machine V that accepts two inputs, a *theorem candidate* T and a witness Π. V reads only a constant number of bits of its input Π and uses only $O(\log |\Pi|)$ random bits. If there is a proof P of T that can be verified by a pointer machine or a RAM in time t, then there is an input Π of size at most $O\left(|t|^{1+\epsilon}\right)$ that will cause V to accept with probability 1. Conversely, if there is an assignment of Π that causes V to accept with probability greater than $1/2$, then C has a satisfying assignment.

Chapter 5

Connections

In this chapter, we will explore some connections between holographic proofs and expander codes. First, we wish to point out that current constructions of holographic proofs lead to codes that have checkers that read only a constant number of bits of their input: Let C be a circuit that is satisfied by every input, and consider the code consisting of the holographic proofs that C is satisfied. If these proofs can be checked by examining a constant number of their bits (as those constructed in Theorem 4.0.1 and Theorem 4.6.1 can be), and if the proofs corresponding to different satisfying assignments have constant relative distance from each other, then the checker of these proofs is a checker in the sense of Definition 4.1.1. This code has a large number of words—one for each satisfying assignment—but it is still highly redundant (even those constructed in Section 4.4 have too much redundancy from the perspective of a coding theorist). For this chapter, we will define a *checkable code* to be a code of constant relative minimum distance and at most polynomial redundancy that has a checker that reads only a constant number of its inputs. All known constructions of checkable codes go through the "PCP-Theorem" (Theorem 4.0.1).

If we examine a construction of a checkable code carefully, we see that the probability that its checker rejects a word is roughly proportional to the distance of that word from the nearest codeword[1]. This rough proportionality is necessary. If a checker examines only

[1] Moreover, the checkable codes that are used in the proof of Theorem 4.0.1 all have this property.

k bits of its input, and if it accepts all codewords, then the probability that it will reject a word of relative distance ϵ from a codeword is at most ϵk. Moreover, we know that there is some constant δ such that if a word has relative distance at least δ from every codeword, then the checker must reject that word with probability at least 1/2. Since the checker only examines k bits of its input, the probability that it rejects a word must change gradually over the space of words.

Expander codes almost have a checker with this property. Consider the following attempt at checking an expander code: choose a few constraints uniformly at random and accept the input word if it satisfies all of them. In the proof of Theorem 2.3.1, we proved that there is a constant δ such that if a word is within distance δ of a codeword, then the probability that this algorithm rejects the word is roughly proportional to its distance from the codeword. However, expander codes are not necessarily checkable. An expander code can have words that are far from any codeword but which satisfy all but one of its constraints.

Consider again the random construction of expander codes in Section 2.3. It is very likely that some constraint will be independent of the others. That means that there are words that satisfy all constraints but this one. Moreover, such a word cannot be close to any codeword, because there can be no word with just one unsatisfied constraint that is close to a codeword. A similar argument shows that if there are two constraints that are not dependent on the others, then there will be words that satisfy all constraints but these two. Thus, if we are going to construct expander codes that are checkable by the means we suggested in this section, then there must be a high level of redundancy among the constraints imposed on the code.

It is interesting to point out that the checkable codes derived from holographic proofs can be decoded by the algorithms used to decode expander codes. Let S be a set of bits that a proof checker might examine. We can form a constraint on the bits in S by saying that the constraint is satisfied by a setting of the bits in S only if the proof checker would accept those bits. The constraints formed by examining all such sets are analogous to the constraints imposed on the expander codes. Moreover, current constructions of holographic proofs induce codes that can be decoded by simple variations of the sequential and parallel expander code decoding algorithms presented in Section 2.3 (these algorithms need to be altered slightly to deal with constraints that are not linear, but the general idea of flipping bits that are in more unsatisfied than satisfied constraints still works).

5.1 Are checkable codes necessary for holographic proofs?

It is conceivable that one could construct holographic proof systems that do not result in checkable codes. Consider the statement of Corollary 4.0.2. This statement is sufficient for most of the results derived from Theorem 4.0.1. Moreover, Corollary 4.0.2 does not say that there must be one holographic proof Π for each satisfying assignment of C or that these holographic proofs must be far apart. Thus, it is not clear that Corollary 4.0.2 leads to a construction of checkable codes.

Arora [Aro94] takes a big step towards showing that statements such as Corollary 4.0.2 actually do imply the existence of checkable codes. He essentially shows that if one is given a holographic proof system in which one can compute in polynomial time a satisfying assignment of a circuit from any holographic proof that the circuit is satisfied, then, assuming that collision-intractable hash functions exist, the set of holographic proofs that a circuit is satisfiable is a code with constant relative minimum distance. All holographic proof systems constructed so far have this property, and it is difficult to imagine one which does not. Arora then shows that any such proof system can be easily modified to yield a checkable code.

In light of Arora's results, we feel that the best hope for finding an alternative construction of holographic proofs is to first try to find a direct construction of checkable codes. One possible way of doing this would be to develop a carefully controlled construction of expander codes in which there is high redundancy among the constraints. We know that it is possible to add a slight amount of redundancy to the constraints, but we do not know how to achieve a constant factor of redundancy. Another advantage of adding redundancy to the constraints is that this would increase the rate of the codes without affecting their error-correction ability. Thus, we feel that the problems of constructing checkable expander codes and better expander codes are intertwined with the problem of finding expander codes with highly redundant constraints.

Bibliography

[ABN$^+$92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–516, March 1992.

[ABSS93] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes and linear equations. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 724–733, 1993.

[AC88] Noga Alon and F.R.K. Chung. Explicit construction of linear sized tolerant networks. *Discrete Mathematics*, 72:15–19, 1988.

[AFWZ] Noga Alon, Uriel Feige, Avi Wigderson, and David Zuckerman. Derandomized graph products. *Computational Complexity*. To appear.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley, Reading, Massachusetts, 1974.

[AKS87] Miklós Ajtai, János Komlós, and Endre Szemerédi. Deterministic simulation in logspace. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 132–139, 1987.

[ALM$^+$92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[Alo86] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[AR94] Noga Alon and Yuval Roichman. Random cayley graphs and expanders. *Random Structures and Algorithms*, 5(2):271–284, April 1994.

[Aro94] Sanjeev Arora. Reductions, codes, PCPs, and inapproximability. Manuscript, November 1994.

[AS92a] Noga Alon and Joel H. Spencer. *The Probabilistic Method.* John Wiley & Sons, New York, 1992.

[AS92b] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1992.

[Bab85] Lászlό Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 421–429, 1985.

[Bab94] Lászlό Babai. Transparent proofs and limits to approximation. In *First European Congress of Mathematics: (Paris, July 6–10, 1992)*, volume 2, pages 31–92. Birkhäuser, 1994.

[BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415, pages 171–176. Springer LNCS, 1990.

[BF93] Lászlό Babai and Katalin Friedl. On slightly superlinear transparent proofs. CS 93-13, The University of Chicago, Department of Computer Science, Chicago, IL, 1993.

[BFL91] Lászlό Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFLS91] Lászlό Babai, Lance Fortnow, Leonid Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 21–31, 1991.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs: applications to approximation. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 294–304, 1993.

[Bie92] Ernst W. Biersack. Performance evaluation of forward error correction in ATM networks. In *Proceedings, ACM SIGCOMM Conference*, pages 248–257, 1992.

[BK89] Manuel Blum and Sampath Kannan. Designing programs that check their work. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 86–97, 1989.

[BLR90] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 73–83, 1990.

[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*, 36:254–276, 1988.

[BoGKW88] M. Ben-or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractibity assumptions. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 113–121, 1988.

[BS94] M. Bellare and M. Sudan. Improved non-approximability results. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 184–193, 1994.

[BW] E. R. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent Number 4,633,470. (filed 1986).

[BZP77] L. A. Bassalygo, V. V. Zyablov, and M. S. Pinsker. Problems of complexity in the theory of correcting codes. *Problems of Information Transmission*, 13(3):166–175, 1977.

[CFLS93] Anne Condon, Joan Feigenbaum, Carsten Lund, and Peter Shor. Probabilistically checkable debate systems and approximation algorithms for pspace-hard functions. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 305–314, 1993.

[CFLS94] Anne Condon, Joan Feigenbaum, Carsten Lund, and Peter Shor. Random debaters and the hardness of approximating stochastic functions. In *Proceedings of the 9th Annual Conference on Structure in Complexity Theory*, pages 280–293, 1994.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms.* The MIT electrical engineering and computer science series. MIT Press, Cambridge, Massachusetts, 1990.

[F9̈4] M. Fürer. Improved hardness results for approximating the chromatic number. manuscript, 1994.

[FGL$^+$91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 2–12, 1991.

[FL92] Uri Feige and László Lovász. Two-prover one-round proof systems: Their power and their problems. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 733–744, 1992.

[FRS88] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. In *Proceedings of the 3rd Annual Conference on Structure in Complexity Theory*, pages 156–161, 1988. Erratum in Structures '90, pp. 318–319.

[Gal63] R. G. Gallager. *Low Density Parity-Check Codes.* MIT Press, Cambridge, MA, 1963.

[GDP73] S. I. Gelfand, R. L. Dobrushin, and M. S. Pinsker. On the complexity of coding. In *Second International Symposium on Information Theory*, pages 177–184, Akademiai Kiado, Budapest, 1973.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–420, 1981.

[GLR$^+$91] P. Gemmell, Richard Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 32–42, 1991.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 291–304, 1985.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, 1989.

[Gol93] Oded Goldreich. A taxonomy of proof systems. *SIGACT News*, 24(4), December 1993.

[Gol94] Oded Goldreich. A taxonomy of proof systems (part 2). *SIGACT News*, 25(1), March 1994.

[GS92] P. Gemmell and M. Sudan. Highly resilient correctors for polynomials. *Information Processing Letters*, 43:169–174, 1992.

[Hei33] Hans Heilbronn. Uber den primzahlsatz von Herrn Hoheisel. *Mathematische Zeitschrift*, 36:394–423, 1933.

[Ima90] Hideki Imai. *Essentials of Error-Control Coding Techniques.* Academic Press, Inc., San Diego, CA, 1990.

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the -29th IEEE Symposium on Foundations of Computer Science*, pages 248–253, 1989.

[Jus76] Jorn Justesen. On the complexity of decoding Reed-Solomon codes. *IEEE Transactions on Information Theory*, 22(2):237–238, March 1976.

[Kah93a] Nabil Kahale. *Exander Graphs.* PhD thesis, M.I.T., September 1993.

[Kah93b] Nabil Kahale. On the second eigenvalue and linear expansion of regular graphs. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 296–303, 1993.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 723–732, 1992.

[Kil94] Joe Kilian. On the complexity of bounded-interaction and noninteractive zero-knowledge proofs. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 466–477, 1994.

[KLR$^+$94] Marcos Kiwi, Carsten Lund, Alexander Russell, Daniel Spielman, and Ravi Sundaram. Alternation in interaction. In *Proceedings of the 9th Annual Conference on Structure in Complexity Theory*, pages 294–303, 1994.

[KU58] A. Kolmogorov and V. Uspenskiĭ. On the definition of an algorithm. *Uspehi Math. Nauk*, 13(4):3–28, 1958. Translation in [KU63].

[KU63] A. Kolmogorov and V. Uspenskiĭ. On the definition of an algorithm. *American Mathematical Society Translations*, 29:217–245, 1963.

[Kuz73] A. V. Kuznetsov. Information storage in a memory assembled from unreliable components. *Problems of Information Transmission*, 9(3):254–264, 1973.

[Lan93] Serge Lang. *Algebra*. Addison-Wesley, 3rd edition, 1993. (material only in 3rd edition).

[Lei92] F. T. Leighton. *Introduction to Parallel Algorithms and Archictectures*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1992.

[LFKN90] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 2–10, 1990.

[Lip91] R. J. Lipton. New directions in testing. In J. Feigenbaum and M. Merritt, editors, *Distributed Computing and Cryptography*, volume 2 of *Dimacs Series in Disc. Math. and Theor. Comp. Sci.*, pages 191–202. A.M.S., 1991.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[LS91] Dror Lapidot and Adi Shamir. Fully parallelized multi prover protocols for NEXP-time. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 13–18, 1991.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.

[Mar73] G. A. Margulis. Explicit constructions of concentrators. *Problemy Peredachi Informatsii*, 9(4):71–80, October-December 1973. English translation available (pp. 325-332).

[Mar88] G. A. Margulis. Explict group theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problems of Information Transmission*, 24(1):39–46, July 1988.

[Mic94] Silvio Micali. CS (computationally sound) proofs. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 436–453, 1994.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North Holland, Amsterdam, 1977.

[Pip77] Nicholas Pippenger. Superconcentrators. *SIAM Journal on Computing*, 6:298–304, 1977.

[Pip93] Nicholas Pippenger. Self-routing superconcentrators. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 355–361, 1993.

[PS77] F.P. Preparata and D.V. Sarwate. Computational complexity of Fourier transforms over finite fields. *Mathematics of Computation*, 31(139):740–751, July 1977.

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly linear-size holographic proofs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 194–203, 1994.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[RS92] Ronitt Rubinfeld and Madhu Sudan. Testing polynomial functions efficiently and over rational domains. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 23–32, 1992.

[Rub90] Ronitt Rubinfeld. *A Mathematical Theory of Self-Checking, Self-Testing and Self-Correcting Programs.* PhD thesis, University of California, Berkeley, CA, 1990.

[Sar77] Dilip V. Sarwate. On the complexity of decoding Goppa codes. *IEEE Transactions on Information Theory*, 23(4):515–516, July 1977.

[Sav69] John E. Savage. The complexity of decoders–part I: Decoder classes. *IEEE Transactions on Information Theory*, 15:689–695, November 1969.

[Sav71] John E. Savage. The complexity of decoders–part II: Computational work and decoding time. *IEEE Transactions on Information Theory*, 17(1):77–85, January 1971.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identies. *J. ACM*, 27:701–717, 1980.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, July 1948.

[Sha49] C. E. Shannon. The synthesis of two terminal switching networks. *Bell System Technical Journal*, 28(1):59–98, January 1949.

[Sha90] Adi Shamir. IP = PSPACE. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 11–15, 1990.

[She91] Shen. Multilinearity test made easy. Manuscript, 1991.

[Spi95] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 1995. To appear.

[SS94] Michael Sipser and Daniel A. Spielman. Expander codes. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 566–576, 1994.

[Sud92] Madhu Sudan. *Efficient checking of polynomials and proofs and the hardness of approximation problems.* PhD thesis, U.C. Berkeley, Oct. 1992.

[Tan81] R. Michael Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, September 1981.

[Tan84] R. Michael Tanner. Explicit construction of concentrators from generalized n-gons. *SIAM Journal Alg. Disc. Meth.*, 5(3):287–293, September 1984.

[vdW53] B. L. van der Waerden. *Modern Algebra*, volume 1. Frederick Ungar Publishing Co., New York, 1953.

[ZP76] V. V. Zyablov and M. S. Pinsker. Estimation of the error-correction complexity of Gallager low-density codes. *Problems of Information Transmission*, 11(1):18–28, May 1976.

[Zuc91] David Zuckerman. Simulating BPP using a general weak random source. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 79–89, 1991.

[Zuc93] David Zuckerman. NP-complete problems have a version that's hard to approximate. In *Proceedings of the 8th Annual Conference on Structure in Complexity Theory*, pages 305–312, 1993.

Index