Reliable Transmission of Information

By Madhu Sudan

1 Introduction

The notion of "digital information" emerged in the middle of the twentieth century, in response to the advent of the telegraph and to the beginnings of computer science, which at the time was principally a theoretical discipline. Of course, the use of electricity to communicate signals goes back further, but the earlier uses involved signals of a "continuous" nature: music, voice, etc. The new era was characterized by the transmission of (or the need to transmit) more "discrete" messages, i.e., messages such as English sentences, which can be described as finite sequences of letters taken from some finite alphabet. The phrase "digital information" came to be applied to such families of messages.

Digital information posed some novel challenges to the engineers and mathematicians charged with the task of communicating such messages. The root cause of these challenges is "noise." Every communication medium is noisy, and never transmits any signal completely accurately. In the case of continuous signals, somehow the receivers (typically, our ears and eyes) can adjust to such errors and learn to discount them. For example, if you play a very old recording of a musical performance, then there will typically be a crackling noise, but it is possible to ignore this, unless the quality is very bad indeed, and concentrate on the music. However, in the case of digital information errors can have a more catastrophic effect. To see this, suppose that we are communicating in English sentences and that the communication medium makes occasional mistakes by altering one of the transmitted letters. In such a scenario the message

WE ARE NOT READY

could easily be changed into the message

WE ARE NOW READY.

All it takes is one error on the part of the communication medium, and the entire intention of the message is reversed. Digital information tends to be inherently intolerant of errors, and the the mathematicians and engineers of the time were charged with the task of inventing methods that would make communication reliable even if the process of transmission is not.

Here is one way of achieving this. To communicate any message, the sender of the message repeats every letter, say five times. For example, to send the message

WE ARE NOT READY

the sender says something like

WWWWWEEEEE AAAAA. . . .

The receiver can then detect errors (as long as there are not too many) by checking that every block of five successive letters repeats the same letter. If this ever fails to be the case, then it is clear that errors have occurred during transmission. If it is not possible for five successive symbols to be in error (or even if it is just very unlikely), then it follows that the resulting scheme is also more reliable than the underlying means of transmission. Finally, if even less error is possible, then it may be possible for the receiver to determine the actual message, rather than simply being able to tell when errors have occurred. For example, if at most two symbols in any block of five can be erroneous, then the most commonly occurring letter in each block of five must be the letter from the original message: a sequence such as

WWWMWEFEEE AAAAA. . .

for instance, would be interpreted by the receiver as

WE A. . . .

Repeating every symbol five times in order to be able to correct two errors does not appear to be a very efficient way to use the communication channel. Indeed, as we will show in the rest of this article, when transmitting long messages one can do much better. However, in order to understand this issue, we need to define the process of communication, the model of error, and the measures of performance more carefully. We do so next.

2 Model

2.1 Channel and Errors

The central object of attention in the problem of information transmission is the "channel of communication," or simply the *channel*. The channel has an *input* (the original signal to be com-

municated) and an *output* (the signal after it is transmitted). The input consists of a sequence of elements from some finite set: by analogy with the English-language example, these elements are called *letters* and the finite set, which is typically denoted Σ, is called an *alphabet*. The channel attempts to transmit the input to the receiver, but while doing so it may make some errors. The alphabet and the process that underlies the errors are what specifies the channel.

The alphabet Σ varies from scenario to scenario. In the example described above, the alphabet consisted of the English characters $\{\mathrm{A}, \mathrm{B}, \dots, \mathrm{Z}\}$, and possibly some punctuation symbols. In most communication scenarios, the alphabet is the "binary alphabet" that consists just of the "letters" 0 and 1, which are known as *bits*. On the other hand, in applications involving storage of digital information (in compact discs (CDs), digital versatile discs (DVDs), etc.), the alphabet contains 256 elements (the alphabet of "bytes").

Specifying an alphabet is easy, but if we wish to define a good mathematical model for the way that errors are produced, then a lot more care is needed. At one extreme is a worst-case model suggested by Hamming (1950), where there is some limit on the number of errors that the channel can make, but within that limit it chooses the errors to be as damaging as possible. A more benign class of errors was proposed by Shannon (1948), who suggested that errors could be modelled by a probabilistic process.

We will focus on one probabilistic model to illustrate many of the concepts below. In this model, the error of the channel is specified by a real number parameter p, where $0 \leqslant p \leqslant 1$. Every use of the channel results in an error with probability p. To be precise, if the sender transmits an element $\sigma \in \Sigma$, then with probability $1-p$ the output for that element is σ but with probability p it is some other element σ' of Σ, chosen uniformly at random. Furthermore, and this is very crucial to this model, the errors are assumed to be *independent*, i.e., the channel repeats this process for each letter it transmits without any memory of how it acted on previous symbols. We refer to this model as the *Σ-symmetric channel with parameter p* (or Σ-SC(p)) in the rest of this article. A special case of particular importance is the *binary symmetric channel*, which is the Σ-symmetric channel when Σ is the binary alphabet $\{0,1\}$. Then, if the input bit is 0, say, the corresponding output bit will be 0 with probability $1-p$ and 1 with probability p.

While this model of error may seem rather oversimplified (and even unnatural if Σ is not the binary alphabet $\{0,1\}$), it turns out that it captures the essence of most mathematical challenges that arise when one tries to make communication reliable. Furthermore, many of the solutions found to make communication reliable in this setting have been generalized to other scenarios, so this simple model is very useful both in practice and in the theoretical study of communication.

2.2 Encoding and Decoding

Suppose the sender wishes to transmit a sequence through a channel that makes errors. One way to compensate for these errors is to send through the channel not the sequence itself but a modified version of the sequence that contains redundant information. The process of modification that we choose is called the *encoding* of the message. We have already seen one method of encoding, namely repeating each term in the sequence several times. However, this is by no means the only way of doing it, so to discuss encoding we use the following general framework: if the sender has a message consisting of a sequence of k elements of Σ, then by some means or another it expands the message into a new sequence, now consisting of n elements of Σ, for some $n > k$. Formally, the sender applies an *encoding function* $E : \Sigma^k \to \Sigma^n$ to the message. (Σ^k stands for the set of sequences of length k with letters in Σ, and Σ^n for the set of sequences of length n.) Thus, to convey a message $m = (m_1, m_2, \dots, m_k)$ to the receiver, the sender transmits over the channel not the k symbols of m but the n symbols of $E(m)$.

The receiver now receives a sequence $r = (r_1, r_2, \dots, r_n)$, belonging to Σ^n, and its goal is to "compress" this sequence back to a k-letter sequence, removing the error and obtaining the original message m (at least if not too many errors have occurred). It does this by applying a *decoding function* $D : \Sigma^n \to \Sigma^k$, which tells it how sequences of length n are converted back into sequences of length k.

The possible pairs of functions E, D describe the options available to the designers of the communication system. Their choice determines the perfor-

mance of the system. Let us now describe how this performance is measured.

2.3 Goals

Very informally, our goals are threefold. We would like to make the communication as reliable as possible. At the same time, we would like to maximize the utilization of the channel. Finally, we would like to do so with effective computation. We describe these goals more carefully below, in the case of the model Σ-SC(p) described earlier.

Consider first the reliability. If we start with a message m, encode it as $E(m)$, and pass it through the channel, then the output, after some random errors have been introduced, will be a string y. The receiver will decode y, producing a new message $D(y)$. For each message m, there is a certain probability of a *decoding error*, i.e., a certain probability that $D(y)$ will not in fact be equal to the original message m. The reliability of the communication is measured by the largest of these probabilities. If this is small, then we know that, whatever the original message m, a decoding error is unlikely, and then we regard the communication as reliable.

Next, let us look at the utilization of the channel. This is measured by the *rate* of the encoding, i.e., the quantity k/n. In other words, it is the ratio of the length of the original message to the length of the encoded message: the smaller this ratio is, the less efficiently one is using the channel.

Finally, practical considerations also require us to be able to encode and decode quickly: a pair of reliable and efficient encoding and decoding functions will not be of much use if they are very time-consuming to compute. Adopting the standard convention in algorithm design, we regard our algorithms as feasible if they run in *polynomial time*: that is, if their running time can be bounded above by a polynomial function of the length of their input and output (see COMPUTATIONAL COMPLEXITY, page **??**, and THE MATHEMATICS OF ALGORITHM DESIGN, page **??**).

To illustrate the above ideas, let us analyze the "repetition encoding" that repeats every letter of the alphabet five times. For simplicity, take the alphabet Σ to be $\{0,1\}$, let the probability p be fixed, and let us consider the behaviour of the model as the message length k tends to ∞. Our encoding function takes strings of length k to strings of length $5k$ and thus has a rate of $\frac{1}{5}$. Given any particular block of five transmissions, the probability that it contains three or more errors is

$$p' = \binom{5}{3}p^3(1-p)^2 + \binom{5}{4}p^4(1-p) + \binom{5}{5}p^5.$$

The probability that that block does not give rise to a decoding error is $1-p'$, so the probability that there is no decoding error is $(1-p')^k$ and the probability that there *is* a decoding error is $1-(1-p')^k$. If we fix $p>0$ and let $k\to\infty$, then $(1-p')^k$ tends to 0 (exponentially quickly), so the probability of decoding error tends to 1. Thus, this encoding/decoding pair is highly unreliable, and its rate is not too good either. The only redeeming feature is that it is very easy indeed to compute. (Its computational efficiency is easily seen to be bounded by a number of operations that is linear in k.)

One way to salvage the repetition code is to repeat every symbol $c\log k$ times. For a largish constant c, the probability of a decoding error goes to 0, but now the rate of the code goes to 0 as well. Prior to the work of Shannon it may have even been believed that a trade-off of this kind was inevitable: every encoding/decoding scheme would either achieve a vanishingly small rate or make mistakes with probability tending to 1. As we will see later in the article, it is in fact possible to define encoding schemes that achieve all three of our goals: they operate at a positive rate, they can correct errors that occur a positive proportion of the time (in either the probabilistic or the worst-case model), and they use efficient encoding and decoding algorithms. Most of the insight for this remarkable result goes back to a seminal paper by Shannon. In that paper he gave the first examples of encoding and decoding functions that satisfied the first two goals, though they were not computationally efficient.

Shannon's encoding and decoding functions were not, therefore, practical, but we can now see, with the benefit of hindsight, that ignoring the goal of efficient computability in order to gain some theoretical insight into the channels was extraordinarily fruitful. A general rule of thumb seems to operate: that the performance of the very best encoding and decoding functions can be matched arbitrarily closely by encoding and decoding functions that are also computationally efficient. This

justifies considering the goal of efficiency separately from the other two goals.

3 The Existence of Good Encoding and Decoding Functions

In this section we will describe results that demonstrate the existence of encoding and decoding functions that have an extremely good rate and reliability. In order to describe these results, first proved by Shannon, it will be useful to consider two related notions introduced by Hamming in work that was essentially concurrent with that of Shannon.

In order to understand these notions, let us start by describing what makes one encoding function E better or worse than another. The task of the *decoding* function is to work out, when it receives a string y, what the original message m was. Notice that this is equivalent to working out what the encoded message $E(m)$ was, since no two messages are encoded in the same way. The possible encoded messages are called *codewords*: in other words, a codeword is a string of length n that arises as $E(m)$ for some message $m \in \Sigma^k$.

What we are worried about is the possibility of confusing two codewords after errors have been introduced, and this depends only on the set of codewords, and not on which codeword corresponds to which original message. Therefore, we adopt what at first seems a strange definition: an *error-correcting code* is any set of strings of length n in the alphabet Σ (that is, any subset of Σ^n). The strings in an error-correcting code are still called codewords. This definition completely ignores the actual process of encoding of a message, but that is so that we can focus on the rate and the decoding error while ignoring computational efficiency. If we are given an encoding function E, then the corresponding error-correcting code is simply the set of all the codewords of E. Mathematically, this is just the image of the function E.

What makes an error-correcting code good or bad? To answer this question, let us consider what happens if the alphabet is $\{0,1\}$ and the code contains two strings $x = (x_1, x_2, \dots, x_n)$ and $y = (y_1, y_2, \dots, y_n)$ that differ in precisely d places. If errors are introduced with probability p, then the probability that x is converted into y is $p^d(1-p)^{n-d}$. Assuming that $p < \frac{1}{2}$, this probability gets smaller as d increases, so the smaller d is, the more likely the strings x and y are to be confused. It seems preferable, therefore, that there should not be too many pairs of strings in the code that differ in just a few places. A similar argument applies to larger alphabets as well.

The above thoughts lead to a definition that is very natural in this context. Given an alphabet Σ and two strings $x = (x_1, x_2, \dots, x_n)$ and $y = (y_1, y_2, \dots, y_n)$ belonging to Σ^n, the *Hamming distance* between x and y is defined to be the number of coordinates i for which $x_i \neq y_i$. For example, let $\Sigma = \{a, b, c, d\}$ and let $n = 6$. The strings *abccad* and *abdcab* differ in the third and sixth places and are identical otherwise, so their Hamming distance is 2. Our goal is to find an encoding function E such that the associated code maximizes the typical Hamming distance between pairs of codewords.

Shannon's solution to this is an extremely simple application of the probabilistic method (see EXTREMAL AND PROBABILISTIC COMBINATORICS on page **??**): he picks the encoding function at random. That is, for every message m, the encoding $E(m)$ is chosen entirely randomly from the set Σ^n, with all choices equally likely. Furthermore, for every message m, this choice is independent of the encoding of every other message m'. It is a good exercise in basic probability to see that such a choice almost always leads to a code where the distances between codewords are on average large. In fact, even the minimum distance between codewords is almost always large. However, we will not show this. Instead, we will argue that with high probability this random choice leads to a "nearly optimal" encoding function, from the point of view of rate and reliability.

First, let us consider what the decoding function ought to be. In the absence of computational requirements, it is not hard to say what the "optimal" decoding algorithm is. If you receive a sequence z, then you should choose the message m that is most likely to have resulted in this sequence. For the model Σ-SC(p) with $p < 1 - 1/|\Sigma|$, it is easily verified that this will be the message m for which the encoding $E(m)$ is nearest to z, as measured by Hamming distance. (If $E(m)$ and $E(m')$ are equal nearest, then one can make an arbitrary choice between them.) The condition on p is impor-

tant here. It ensures that when the sequence $E(m)$ passes through the channel, the most likely output corresponding to any given term, out of the $|\Sigma|$ different possibilities, is the same as the input. Without this condition, there would be no reason to expect z to be close to $E(m)$. We shall argue that there is a number C, depending only on the error probability p and the size of the alphabet, such that for a random encoding function with rate smaller than C, this decoding function recovers the original message with a high probability. As an aside, Shannon also showed that for the same constant C, any attempt to communicate at rates greater than C would lead to errors with probability exponentially close to 1. Because of this result, the constant C is known as the *Shannon capacity* of the channel.

Once again, for simplicity we shall consider just the case of the binary alphabet $\{0,1\}$. In this case we are choosing a random function E from $\{0,1\}^k$ to $\{0,1\}^n$, and we would like to show that, under suitable circumstances, the resulting code will almost certainly be very reliable. In order to do this, we shall focus on a single message m, and rely on two basic ideas.

The first idea is a precise form of the law of large numbers. If the error probability is p, then the expected number of errors introduced into a codeword $E(m)$ is pn, so, if n is large, then we expect that the actual number of errors will almost certainly be very close to this, just as, if you toss a fair coin 10 000 times, you will be surprised if the number of heads is not close to 5000. The result that expresses this formally is as follows.

Claim. There exists a constant $c > 0$ such that the probability that the number of errors exceeds $(p+\epsilon)n$ is at most $2^{-c\epsilon^2 n}$.

The same can be said of the probability that the number of errors is less than $(p-\epsilon)n$, but we shall not use this result.

When n is large, $2^{-c\epsilon^2 n}$ is extremely small, so the number of errors is almost certainly at most $(p+\epsilon)n$. The number of errors equals the Hamming distance from y, the output of the channel, to $E(m)$, the codeword that was transmitted. Therefore, the decoding algorithm that chooses the codeword with smallest Hamming distance from y will almost certainly choose $E(m)$, provided that there is no message m' such that $E(m')$ is closer to y than $(p+\epsilon)n$.

The second idea, which allows us to say that this will almost certainly be the case, is that "Hamming balls are small." Let z be a sequence in $\{0,1\}^n$. Then the *Hamming ball of radius r about z* is the set of all sequences w with Hamming distance at most r from z. How big is this set? Well, in order to specify a sequence w with Hamming distance exactly d from z, it is enough to specify the set of d places where w and z differ. There are $\binom{n}{d}$ ways of choosing this set, so the number of sequences with distance at most r is

$$\binom{n}{0}+\binom{n}{1}+\binom{n}{2}+\cdots+\binom{n}{r}.$$

If $r=\alpha n$ and $\alpha<\frac{1}{2}$, then this number is at most a constant times $\binom{n}{r}$, because each term is at least

$$\frac{n-r}{r}=\frac{1-\alpha}{\alpha}$$

times the one before. But

$$\binom{n}{r}=\frac{n!}{r!(n-r)!}.$$

If we now use Stirling's formula (see THE GAMMA FUNCTION on page **??**) or the looser approximation $n!=(n/\mathrm{e})^n$, then we find that this is about $(1/\alpha(1-\alpha))^n$, which is $2^{H(\alpha)n}$, where

$$H(\alpha)=-\alpha\log_2\alpha-(1-\alpha)\log_2(1-\alpha).$$

(Note that $H(\alpha)$ is positive, because α and $1-\alpha$ are less than 1 and therefore have negative logarithms.) The function H is called the *entropy function*. It is continuous and strictly increasing on the interval $[0,\frac{1}{2}]$ with $H(0)=0$ and $H(\frac{1}{2})=1$. So, if $\alpha<\frac{1}{2}$, then $H(\alpha)<1$, and therefore $2^{H(\alpha)n}$ is exponentially smaller than 2^n: this is what is meant by saying that the Hamming ball of radius αn is small.

Let us set α to be $p+\epsilon<\frac{1}{2}$. Then the probability that a single randomly chosen sequence $E(m')$ lies in the Hamming ball of radius $(p+\epsilon)n$ about y is at most $2^{H(p+2\epsilon)n}2^{-n}$. (The 2ϵ is to compensate for slight inaccuracies in the above estimate for the size of the ball.) Since there are 2^k-1 possibilities for m', the probability that one can be found for which $E(m')$ lies in the ball is at most $2^k 2^{H(p+2\epsilon)n}2^{-n}$. Therefore, if $k\leqslant n(1-H(p+$

$2\epsilon) - \epsilon)$, this probability is at most $2^{-\epsilon n}$, which is exponentially small.

Because we can choose ϵ to be as small as we like, we can make k/n as close as we like to $1 - H(p)$ while still maintaining an exponentially small probability of decoding error. It turns out that the quantity $1 - H(p)$ is the constant C discussed earlier: the Shannon capacity of the binary symmetric channel. Thus, the capacity of the binary symmetric channel is always positive if $p < \frac{1}{2}$.

Shannon's theorem and proof are significantly more general than the above example demonstrates. For a wide variety of channels, and for a wide variety of models of (probabilistic) error, his theory pins down the capacity of the channel and shows that reliable communication is possible if and only if the rate of the channel is less than its capacity. Shannon's proof is a remarkable example of the use of the probabilistic method in the practice of engineering. Note, however, that the encoding and decoding algorithms are quite impractical. The proof gives no clue about how to find an encoding function, though of course one can consider every encoding function $E : \{0,1\}^k \to \{0,1\}^n$ to check if it is good. However, even if such a function is found, it may have no succinct description, in which case the encoder and decoder have to store this encoding function as an exponentially long table in their memory. Finally, the decoding algorithm seems to involve a brute-force search for the nearest codeword, a problem which seems to be the most serious obstacle to obtaining a computationally efficient version of Shannon's theorem that can be used in practice. What the theorem definitely *does* give us is a significant insight into the limitations and potential utility of the communication channel. With this in mind, we can set ourselves the right targets to strive for when we come to devise more practical encoding and decoding procedures. In the next section we will show that it is possible to achieve a fixed rate that is bounded away from zero, to tolerate a constant fraction of errors, and to do both of these with efficient algorithms.

4 Efficient Encoding and Decoding

We now turn to the task of designing encoding and decoding functions that can be calculated efficiently. Currently, there are at least two very different approaches to building such functions. We describe here an approach based on algebra over finite fields. The alternative approach is based on the construction of EXPANDING GRAPHS, but we will not describe that here.

4.1 Codes for Large Alphabets Using Algebra

In this section we describe a simple way to get an encoding function $E : \Sigma^k \to \Sigma^n$, where Σ is a finite FIELD with at least n elements. (Recall that there are finite fields with q elements whenever q is of the form p^t for a prime p and a positive integer t.) These codes were introduced by Reed and Solomon (1960) and have since been called the *Reed–Solomon codes*.

A Reed–Solomon code is specified by a sequence of n distinct field elements $\alpha_1, \dots, \alpha_n \in \Sigma$. Given a message $m = (m_0, m_1, \dots, m_{k-1}) \in \Sigma^k$, we associate with the message the polynomial $M(x) = m_0 + m_1 x + \cdots + m_{k-1}x^{k-1}$. The encoding of m is simply the sequence $E(m) = M(\alpha_1), M(\alpha_2), \dots, M(\alpha_n)$. In other words, to encode a sequence m, you treat the terms of the sequence as the k coefficients of a polynomial of degree $k-1$ and write out the values that this polynomial takes at $\alpha_1, \dots, \alpha_n$.

Before describing the error-correcting capability of this code, let us note that it is very succinctly represented: all that is needed to specify it is a description of the field Σ and the sequence of n elements $\alpha_1, \dots, \alpha_n$. It is easy to show that the number of additions and multiplications needed to compute $M(\alpha)$ is at most Ck for some constant C. (For example, to work out $3\alpha^3 - \alpha^2 + 5\alpha + 4$, you start with 3, multiply by α, subtract 1, multiply by α, add 5, multiply by α, and add 4.) Therefore, the number of field operations needed to compute the entire encoding is bounded above by Cnk, for some (different) constant C. (In fact, more sophisticated and efficient algorithms are known for the encoding problem that take at most $Cn(\log n)^2$ steps.)

Now let us consider the error-correcting properties of the code. We start by showing that the encodings of any two messages m_1 and m_2 have a Hamming distance of at least $n - (k-1)$. To see this, let $M_1(x)$ and $M_2(x)$ be the polynomials associated with m_1 and m_2. Now the difference $p(x) = M_1(x) - M_2(x)$ has degree at most $k-1$,

and it is not the zero polynomial (since M_1 and M_2 are distinct), and therefore it has at most $k-1$ roots. This tells us that there are at most $k-1$ values of α for which $M_1(\alpha) = M_2(\alpha)$. It follows that the Hamming distance between the sequences

$$E(m_1) = (M_1(\alpha_1), M_1(\alpha_2), \dots, M_1(\alpha_n)$$

and

$$E(m_2) = (M_2(\alpha_1), M_2(\alpha_2), \dots, M_2(\alpha_n)$$

is at least $n-k+1$.

It follows that if z is any sequence, then its Hamming distance from at least one of $E(m_1)$ and $E(m_2)$ is greater than $\frac{1}{2}(n-k)$ (since otherwise the distance between $E(m_1)$ and $E(m_2)$ would have to be at most $n-k$). Therefore, if the number of errors that occur during transmission is at most $\frac{1}{2}(n-k)$, then the original message m is uniquely determined by the received sequence z. What is much less obvious is that there is an efficient algorithm for working out what m was, but, remarkably, it is possible to compute m with a polynomial-time algorithm (in n), which we shall now describe.

What must the decoding algorithm do? It is given the numbers $\alpha_1, \dots, \alpha_n$ and the received sequence $z_1, \dots, z_n$, and is required to find a polynomial M of degree $k-1$ or less such that $M(\alpha_i) = z_i$ for all but at most $\frac{1}{2}(n-k)$ values of i. If such a polynomial exists, then it is unique, as we have just seen, and its coefficients will give the original message m (if the number of errors is at most $\frac{1}{2}(n-k)$).

If there were no errors, then our task would be much easier: one can determine the coefficients of a polynomial of degree $k-1$ from k of its values by solving k simultaneous equations. However, if some of the values we use are incorrect, then we will end up with a completely different polynomial, so this method is not easy to use for the problem we actually face.

To overcome this difficulty, let us imagine that M exists and that the errors introduced into the sequence $M(\alpha_1), \dots, M(\alpha_n)$ occur at $i_1, \dots, i_s$, where $s \leqslant \frac{1}{2}(n-k)$. Then the polynomial $B(x) = (x-\alpha_{i_1}) \dots (x-\alpha_{i_s})$ has degree at most $\frac{1}{2}(n-k)$ and is zero if and only if x is equal to α_{i_j} for some j. Let us set $A(x)$ to equal $M(x)B(x)$. Then $A(x)$ is a polynomial of degree at most $k-1+\frac{1}{2}(n-k) = \frac{1}{2}(n+k-2)$, and for every i we have $A(\alpha_i) = z_i B(\alpha_i)$. (If there is no error at i, then this is obvious, since $z_i = M(\alpha_i)$, and if there is an error at i, then both sides are 0.)

Conversely, suppose that we manage to find polynomials $A(x)$, of degree at most $\frac{1}{2}(n+k-2)$, and $B(x)$, of degree at most $k-1$, such that $A(\alpha_i) = z_i B(\alpha_i)$ for every i. Then $R(x) = A(x) - M(x)B(x)$ is a polynomial of degree at most $\frac{1}{2}(n+k-2)$, and $R(\alpha_i) = 0$ whenever $M(\alpha_i) = z_i$. Since there are at most $\frac{1}{2}(n-k)$ errors, this happens for at least $n - \frac{1}{2}(n-k) = \frac{1}{2}(n+k)$ values of i. Therefore, the number of roots of R is bigger than its degree, from which it follows that R is identically zero, so that $A(x) = M(x)B(x)$ for every x. From this we can determine M: given k values of x for which $A(x)$ and $B(x)$ are nonzero, one can determine k values of $M(x) = A(x)/B(x)$, and hence determine M.

It remains to show that we can indeed (efficiently) find polynomials $A(x)$ and $B(x)$ with the required properties. The n constraints $A(\alpha_i) = z_i B(\alpha_i)$ turn into n linear constraints on the unknown coefficients of A and B. Since B has $\frac{1}{2}(n-k)+1$ coefficients and A has $\frac{1}{2}(n+k)$ coefficients, the total number of unknowns is $n+1$. Since the system of equations is homogeneous (that is, we obtain a solution if we take all unknowns to be zero) and the number of unknowns is greater than the number of constraints, there must be a nontrivial solution: that is, a solution where $A(x)$ and $B(x)$ are not both the zero polynomial. Moreover, we can find such a solution by Gaussian elimination, which takes at most Cn^3 steps.

To summarize: we construct a code by exploiting the fact that two distinct low-degree polynomials cannot be equal for too many values. We then exploit the rigid algebraic structure of low-degree polynomials for the purposes of decoding. The main tool that allows us to do this is LINEAR ALGEBRA and in particular the solving of systems of simultaneous equations.

4.2 Reducing the Size of the Alphabet Using Good Codes

The ideas described in the previous section show us how to build codes with efficient encoding and decoding algorithms, but they use relatively large alphabets. In this section we shall exploit these results to build binary codes.

To begin with, let us consider a very obvious method of converting codes over large alphabets into codes over the binary alphabet $\{0,1\}$. For simplicity, assume that we have a Reed–Solomon code over an alphabet Σ of size 2^ℓ for some integer ℓ. Then we can associate the elements of Σ with binary strings of length ℓ. In such a case, we can regard the Reed–Solomon encoding function, which maps Σ^k to Σ^n, as a function from $\{0,1\}^{\ell k}$ to $\{0,1\}^{\ell n}$. (For instance, an element of Σ^k is a sequence of k objects, each of which is a binary sequence of length ℓ. Putting them together produces a single binary sequence of length $k\ell$.) Since the encodings of two distinct messages differ for at least $n-k+1$ elements of Σ, they must also differ on at least $n-k+1$ bits.

This gives a fairly reasonable code over the binary alphabet. However, $n-k+1$ is not as large as a fixed fraction of ℓn: the ratio $(n-k+1)/\ell n$ is less than $1/\ell$, and since we need 2^ℓ, the size of Σ, to be at least n, we find that this fraction is at most $1/\log_2 n$, which tends to zero as n tends to infinity. However, this can be fixed in a simple way, as we shall see.

The problem with the simple binary approach is that two different elements of Σ may be represented by binary sequences that differ in just one bit. However, the Hamming distance between two binary sequences of length ℓ is usually much larger: it is more like $c\ell$ for some positive constant c. Suppose that we could represent the elements of Σ as binary sequences of some length L in such a way that the Hamming distances between any two of the sequences used was at least cL. This would allow us to improve our argument above: if the encodings of two messages were different for at least $n-k+1$ elements of Σ, then they would have to differ on at least $cL(n-k+1)$ bits rather than just $n-k+1$, and this is a positive fraction of Ln.

What we are asking for is an encoding of the binary sequences of length ℓ as sequences of length L in such a way that no two codewords are closer than cL to each other. But we know, from the previous section, that such an encoding exists, provided that L and c satisfy appropriate conditions: for instance, it is possible to find an encoding function that works with $L \leqslant 10\ell$ and $c \geqslant \frac{1}{10}$.

So how do we use this? We start with a binary sequence m of length ℓk. As above, we associate with this a sequence of length k in the alphabet Σ. We then encode this sequence using the Reed–Solomon code, obtaining a sequence of length n in the alphabet Σ. Next, we convert each term of this sequence into a binary sequence of length ℓ. And, finally, we encode each of these n binary sequences as a sequence of length L using a good encoding function, obtaining as a result a binary sequence of length Ln. We then pass this sequence through the channel, where errors may be introduced. The receiver then breaks the received sequence up into n blocks of length L, decodes each block to work out what binary sequence of length ℓ gave rise to it, and interprets that binary sequence as an element of Σ. This results in a sequence of n elements of Σ. It then uses the Reed–Solomon decoding algorithm to decode this sequence, producing a sequence of k elements of Σ. Finally, this can be converted into a binary sequence of length ℓk.

We have said nothing about the efficiency of the encoding and decoding procedures that convert binary sequences of length ℓ into ones of length L and back again, stating merely that they exist. Since efficiency is supposed to be our priority, this may seem rather strange: do we not now face exactly the same problem that we were trying to solve in the first place? Luckily we do not, because although these encoding and decoding procedures may take exponentially long, they take exponentially long as a function of L, and L is much much smaller than n. Indeed, L is proportional to $\log n$, from which it follows that 2^L is bounded above by a polynomial function of n. This is a useful principle: one can afford procedures of exponential complexity provided that one only ever applies them to very short strings.

Thus even though we have not managed to specify the code explicitly, we have demonstrated that there is an encoding and decoding algorithm that runs in polynomial time and that corrects a constant fraction of errors. To complete this section, let us address the question of the probability of decoding error, which we have not yet discussed. The technique described above, of composing encoding functions (and decoding functions), can also be used to improve the above code so that the encoding and decoding still take place in polynomial time, but now the decoding error probability is exponentially small on the binary symmetric channel with parameter p, and the rate is arbi-

trarily close to the Shannon capacity, which is the theoretical maximum. (The idea is to compose a Reed–Solomon code that has rate close to 1 with a random inner code, and then to show that with random errors most of the inner decoding steps decode correctly. One then uses the outer decoding step to convert the "mostly correct decoding" to a "fully correct decoding.")

5 Impact on Communication and Storage

The mathematical theory of error-correcting codes has made a deep impact on the technologies for storage and communication of information, and we elaborate a little on this below.

Storage of information on digital media is probably the biggest success story for error-correcting codes. Most known forms of storage media, and in particular standards for audio and data CDs and DVDs, prescribe error-correcting codes based on Reed–Solomon codes. Specifically, they are based on a code that maps $\mathbb{F}_{256}^{223}$ to $\mathbb{F}_{256}^{255}$, where $\mathbb{F}_{256}$ is the finite field with 256 elements. In audio CDs, codes are use to protect from minor scratches, though more serious scratches do lead to audible errors. In data CDs the error correction is stronger (with more redundancies), so that even serious scratches do not lead to loss of data. In all cases (CDs and DVDs) the readers for these devices use fast algorithms for decoding when reading the information on the media. Typically, these algorithms are based on the idea of the previous section, but are much faster implementations (in particular, an algorithm due to E. Berlekamp is widely used). Indeed, several CD readers owe their faster reading speed to faster decoding algorithms. Similarly, the increased storage capacity of DVDs (compared with CDs) is attributed in part to better error-correcting codes. Indeed error-correction technology played a crucial role in establishing the dominance of audio CDs, which store music digitally, over the classical, and now almost extinct, gramophone records, which store music in continuous forms. Thus, mathematical advances in coding theory have played an influential role in this technology.

Similarly, error-correcting codes have had a profound effect on communication. Since the late 1960s, error-correcting codes (and decoding) have been used for communication from satellites to their base stations on Earth. Of late, error-correcting codes are also being used in cellular phone communications, and modems. Again, the most commonly used code at the time of the writing of this article is the Reed–Solomon code, though this situation has been changing rapidly since the discovery of a new class of codes called "turbo codes." This new family of codes seems to offer significant resilience to random errors (more so than that offered by methods based on Reed–Solomon codes) and uses a simple and quick algorithm, even when the codes used have small block length. These codes and the corresponding decoding algorithm have led to a resurgence of interest in codes constructed with the help of insights from graph theory (see Graphs on page **??**). Many of the good properties of turbo codes have been observed only empirically: that is, the codes seem to work very well in practice but it has not yet been proved rigorously that they do. Nevertheless, the observations have been so compelling that new standards for communication are starting to prescribe these codes.

Finally, it must be stressed that while many of the codes used are based on ones that are studied in the mathematical literature, this should not be taken to mean that they can be deployed immediately without further design. For example, the Mariner spacecraft used not a Reed–Muller code but a variant of it designed to allow for synchronization between blocks. Similarly, the Reed–Solomon codes used in storage devices are carefully spread out over the disc, so as to allow the physical device to resemble more closely the model of a code over a large alphabet. Note that errors due to a scratch on the disc surface tend to ruin a large collection of bits in a small localized part of the disc. If all the data from a block were sitting in such a neighborhood, the entire block would be lost. So each block of 255 bytes of information is spread out all over the disc. On the other hand, the bytes themselves, which are elements of $\mathbb{F}_{256}$, are written as eight bits in close proximity. So a scratch corrupting one bit out of these eight is also likely to corrupt others in the neighborhood. However, this is all right from the perspective of the model that views the entire collection of eight bits as a single element. In general, working out the right way to apply the theory of error correction to a given scenario is a major challenge, and many

success stories would not have been success stories had it not been for some careful design choices.

Mathematics and engineering continue to feed each other in this arena. Mathematical successes, such as new algorithms for decoding Reed–Solomon codes, raise the challenge of how to adapt technology to exploit new algorithms. Engineering successes, such as the discovery of turbo codes that perform extremely well, challenge mathematicians to come up with a formal model and analysis that can explain this success. And if such a model and analysis emerges, it is likely to lead to the discovery of new codes that might surpass the performance of turbo codes and lead to a new set of standards!

6 Bibliographic Notes

The theory of reliable communication and storage of information owes much to the seminal works of Shannon (1948) and Hamming (1950), which formed the basis for much of this article. The Reed–Solomon codes of Section 4.1 are from Reed and Solomon (1960). Their decoding algorithm originates in the work of Peterson (1960), though the algorithm given here is significantly simplified. The technique of composing codes is due to Forney (1966).

Over the years, coding theory has amassed a wide variety of results. Some of these give better constructions of codes with faster algorithms. Others provide theoretical upper limits on how well codes can perform. The theory uses an enormous variety of mathematical tools, many of them more advanced than the ones described in this article. Most notable among them are algebraic geometry and graph theory, which are used to construct very good codes, and the theory of orthogonal polynomials (see TITLE OF ARTICLE AS-YET UNDECIDED on page **??**), which is used to prove limits on parameters of codes, such as their rate and reliability. Most of the highlights of this vast literature are covered in Pless and Huffman (1998).

Further Reading

Hamming, R. W. 1950. Error detecting and error correcting codes. *Bell System Technical Journal* 29:147–160.

Forney Jr, G. D. 1966. *Concatenated Codes*. Cambridge, MA: MIT Press.

Peterson, W. W. 1960. Encoding and error-correction procedures for Bose–Chaudhuri codes. *IEEE Transactions on Information Theory* 6:459–470.

Pless, V. S. and W. C. Huffman (eds). 1998. *Handbook of Coding Theory* (two volumes). Amsterdam: North-Holland.

Reed, I. S. and G. Solomon. 1960. Polynomial codes over certain finite fields. *SIAM Journal of Applied Mathematics* 8:300–304.

Shannon, C. E. 1948. A mathematical theory of communication. *Bell System Technical Journal* 27:379–423, 623–656.