ROBERT W. LUCKY
SILICON DREAMS:
INFORMATION, MAN,
AND MACHINE
NY: ST. MARTIN'S, 1989

2

A THEORY OF INFORMATION

INFORMATION THEORY

Information theory. When I was a graduate student it sounded like an all-encompassing theory of nature. Perhaps it has since fallen short of the expectations of my callow youth, but it still fascinates me. Most great physical and mathematical discoveries seem trivial after you understand them. You say to yourself, "I could have done that." But as I hold the tattered journal containing Claude Shannon's classic 1948 paper, "A Mathematical Theory of Communication," I see yellowed pages filled with vacuum tubes and mechanisms of yesteryear, and I know that I could never have conceived the insightful theory of information shining through these slippery, glossy pages of archaic font. I know of no greater work of genius in the annals of technological thought.

The period of time immediately following World War II was incredibly ripe for technology. The wartime impetus behind radar led to microwave radio transmission. The need for secrecy gave rise to digital transmission formats and pulse code modulation, which began the digital revolution in communications that still continues today. Information theory, the digital computer, and the transistor were all invented in these few short years. If the theory of punctuated equilibrium—that is, evolution in stepwise bursts—can be

applied to the history of technology, this fertile period would appear to be the perfect example of such an evolutionary burst.

Shannon himself was able to indulge his growing interest in deriving a theoretical formulation for information through his wartime work in cryptography. As a boy he had been captivated by Edgar Allan Poe's "The Gold Bug," a story that has fascinated the youth of several generations. In his college years at Michigan and MIT, Shannon had read a 1928 paper by R. V. L. Hartley (the same Hartley, incidentally, who invented the best-known vacuum tube oscillator) dealing with the transmission of information, which Shannon said later had been an important influence on his life. At Bell Labs in the years between 1940 and 1945 Shannon began where Hartley had left off, working on information and communications and using the application to cryptography as a way of legitimatizing his work. The first mention of the phrase "information theory" occurs in a 1945 memorandum entitled "A Mathematical Theory of Cryptography." Curiously, this phrase never is used in his famous 1948 paper, which became the cornerstone of the new field of information theory.

Shannon's information theory is a philosophy of information from the point of view of communications. It is seldom prescriptive. It gives us a mathematical measure of information and of the informational capacity of a communications channel. Its central result is a theorem about the transmission of information over a communications channel, a result that has served as an inspiration to communications designers now for almost half a century. Shannon's genius lay in exposing a new way of thinking about information and communication. He pointed in a direction and set out the bounds within which one must stay. By now the road has been well traveled. The *Transactions on Information Theory,* the principal journal in the field, is filled with mathematical, abstract papers on esoteric problems with long titles—indicative of the degree of specialization that has been reached.

Information theory is primarily concerned about the *mechanics* of information handling. In the introduction to Shannon's paper we find the following:

> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem.

Thus, for the most part, information theory is not concerned with the meaning or the value of the information it describes. A bit of information could represent the chance throw of a coin—a head or a tail—or in the World War II era it could have represented the choice of location of the Allied invasion of France, whether it would be the coast of Normandy or Calais. In the eyes of information theory the same amount of information would be involved in either case, that is, the same storage capacity or the same transmission capacity would be required. Surely, you might argue, there is a great deal of difference. The chance flip of a coin might be of no consequence whatsoever, yet Hitler's foreknowledge of the location of the Allied invasion might have changed the course of the world.

In some respects I would agree that the disregarding of what Shannon called "meaning" is a philosophical deficiency of information theory. Yet I would despair of producing useful concepts in a theory that accounted for a property that is as difficult to measure as meaning, or, what is a consequence, value. Information theory considers information in much the same sense that we might study money in terms of the size and weight of the paper on which it is printed. Using such a theory we might derive the size of truck needed to transport our currency or the vault space required for its storage, but we would not be concerned with the fluctuations of the exchange rate or the effects of inflation on the intrinsic value of our paper certificates. Nevertheless, we shall see that Shannon's theoretical viewpoint provides deep insight about the generation and interpretation of information—insight that often borders on the questions of meaning and value.

MEASURING INFORMATION

How shall we measure information? This is a question not far removed from asking what information is, but nowhere in this

entire book will I give a definition of "information." It is a term so much used for so many different purposes that I have found that I cannot avoid using it in various ways myself. In the classical, Shannon sense, however, information has a precise, mathematical interpretation that I should like to introduce by considering intuitively what it means to have more or less of this quantity we call information.

Perhaps the cornerstone of information theory is the observation that any would-be information that we receive is a combination of the known and the unknown. Because it is so important, let me say synonymously that information is a blend of the expected and the unexpected, the foreseen and the unforeseen, or the certain and the uncertain. The first part, the expected, carries no real information; the measurable information is contained in the latter part, the unexpected.

If I tell you something that you already know, I give you no information. After all, it would seem that the whole idea of information is to "inform." More generally, even if I tell you something that you can deduce from other things that you know, I give you no information. There is nothing new added to your store of knowledge. The essence of information lies in the newness—the differentness—from that previously known, and from that which may be expected based upon what is previously known.

If I tell you something and your response is a shrug of the shoulders and the comment, "I figured as much," then little information has been conveyed. On the other hand, if your natural reaction is a surprised, "Wow, I had no idea!" then it would seem that you had received a significant amount of information. The measure of information intuitively must lie in the amount of surprise that resides in the unexpected or uncertain portion of the data received. We take the essence of information as the irreducible, fundamental underlying uncertainty that is removed by its receipt. In the Shannon sense information is measured through uncertainty. Information is viewed as being the *resolution of uncertainty.*

As the unit of information we take the prototypical uncertain event, the single throw of a fair coin. There are two possibilities that are equally likely, and we have no expectations, no prior indications, of whether the toss will yield a head or a tail. The actual throw resolves this uncertainty and gives us what we define as one bit of information. A bit resolves a single, binary choice. On a computer

it is represented by a 1 or a 0. In everyday life it is the answer to a single "yes or no" question.

The word "bit" itself, which is a contraction of "binary digit," was invented by statistician John Tukey during a lunchtime conversation at Bell Labs. Although Tukey has other significant claims to a well-deserved fame, such as the popular and important Fast Fourier Transform, this little word grants him a kind of immortality in my mind. Imagine being the *inventor* of a word that is being used so constantly throughout the world! But let us return to our inspection of its deeper meaning.

So a single bit resolves the uncertainty of a coin toss. It is a far cry from that simple definition to the everyday information that we receive in so many different forms and formats. Life is not exactly constructed as the outcome of a series of coin tosses. However, if we are to measure information in the Shannon sense, we must interpret it in exactly this way. How many coin flips does it take to resolve the uncertainty in a given case? This is the measure of the information received.

There is an equivalent and more constructive way of getting at the number of "information bits" in data that we receive. It is also the *minimum number of yes/no questions required on the average to arrive at the given data,* knowing everything that we do know. Each bit of information is the answer to a single yes/no question, posed according the best possible strategy for the given situation. If you did not have this data, how many yes/no questions would it take you to reconstruct it?

This minimum average number of yes/no questions has a very real significance. It is not just a mathematical abstraction, for it is clearly the minimum amount of computer storage in bits required to save the data. Having the answers to the series of yes/no questions allows us to rederive the data any time we need it. This number is also directly related to the communications capacity required to send the information to us. How many bits have to be transmitted? Finally, this number may well be related to the human capability to intake, memorize, and output the data. The number of bits of information is the fundamental uncertainty associated with it.

Beginning with the single throw of a coin, I want to consider intuitively a progression of more complicated informational events. First let us change from a coin to a roulette wheel with 32 numbers. (It is a little smaller than usual for numerical convenience.) Again

you have no prior expectations, and all 32 numbers represent equally likely outcomes of a single roll of the ball. How much information is required to convey an outcome? Well, it does not take too much cleverness to think of a good strategy for asking a series of yes/no questions to pin down a given result. "Is it sixteen or under?" you ask. If the answer is yes, you divide the range in half, and ask whether or not it is, for example, 8 or under. You continue to divide the range by two until you arrive at the number. It takes exactly five questions. There is no strategy that on average works better for this situation, so the information required to store or transmit a sequence of results of the roulette wheel is 5 bits per outcome.

We must be clear that the strategy for asking yes/no questions must yield the least questions *on average.* You might be tempted, for example, to ask immediately, "Did the ball land on number 22?" (You feel lucky.) I might look up in surprise, and say yes. "How did you know?" I ask in consternation. You smile knowingly and feel that you have beaten the system, that perhaps it really took only one bit to learn this particular information. But most of the time—$\frac{31}{32}$, to be exact—you will have wasted one precious question and gotten very little in return. That would be a bad strategy. The information content is represented by the strategy that works best on the average. To get at the measure of information you must imagine this exact situation being enacted many, many times—preferably by your clones in parallel universes—while the intrinsically random portion of the data varies over all possible cases according to its underlying probability distributions. Your strategy must enable you to specify the data in the fewest questions on the average over all of these enactments.

In the case of the 32-number roulette the branching strategy requires 5 questions, so the measure of the information associated with an outcome is 5 bits. If you have to save a sequence of roulette outcomes on your computer, you will have to devote an average of 5 bits to each. (In fact, you would use exactly 5 bits on each.) Notice that 5 is the number of times that 32 can be divided in half, or in mathematical notation $\log_2 32 = 5$. In any similar situation with equally likely outcomes the information associated with the result would be the logarithm (to the base 2) of the number of possible outcomes. A 64-number roulette wheel would produce a series of numbers that would take 6 bits apiece to store.

Where there are n equally likely outcomes, the information is

$\log_2 n$. (There are only a few more logarithms in this book, but in all cases they will be understood to be to the base 2.) The actual probability of getting as an outcome any individual value is of course $\frac{1}{n}$, so we could as well specify the information as

$$\text{information} = -\log_2(\text{probability}).$$

Notice that the information is positive, since the logarithm of a number less than one, such as a probability, is negative. This simple expression has profound meaning when properly interpreted. It says that the information is measured by the logarithm of the probability. The lower the probability of an event, the more surprising it is, and the more information it represents, but we must scale by the logarithm. If an event is half as likely as another event, for example, it gives us one more bit of information.

Although we have only shown that this simple relationship measures the information associated with a set of equally likely outcomes, Shannon proved that it is indeed a very general measure, that no other measure satisfies the basic requirements that the information always increases as the probability decreases (which is the element of "surprise"), and that the sum of two informational events has an information measure that is the sum of their individual information measures. (The probability of two independent events is the product of their individual probabilities; in logarithms, which scale the information measure, this is a sum.) Thus information is simply measured by the logarithm of the probability. The only problem in application, and it is a monstrous one, is that we must be able to enumerate all possible situations, evaluate the probability of each given all prior information, and average the information measure of all of these possibilities.

EXAMPLES OF INFORMATION

Now let me complicate the situation just slightly. Suppose that you place a bet on a particular number on the roulette wheel. The information you want now is not the winning number, which as we have seen requires 5 bits to specify, but merely whether or not you

have won or lost. How much information does that take? Obviously here the intrinsic uncertainty is less than before; you do not expect to win. Most of the time I give you the uninteresting news that you have lost. A little of the time I give you the unexpected, information-rich, and incidentally happy news that you have won.

By analogy with the previous discussion we might guess that the information is the probability-weighted average of the information associated with each of the two possible outcomes. That is, $\frac{1}{32}$ of the time we get $-\log\frac{1}{32}$ bits of information, and $\frac{31}{32}$ of the time we get $-\log\frac{31}{32}$ bits.

$$\textit{Information} = -(\tfrac{31}{32})\times\log(\tfrac{31}{32}) - (\tfrac{1}{32})\times\log(\tfrac{1}{32}) = 0.201 \textit{ bits.}$$

This is, in fact, the information associated with the news as to wins and losses in our bets. If we were to store in our computer a sequence of wins and losses, such as

L L L L L L L L W L L L L L L L L L L L L . . .

this simple relation tells us that it requires on the average about $\frac{1}{5}$ of a bit per outcome. Turning this news around, it means that there is a strategy for asking questions about these results that can specify the wins and losses with about an average of $\frac{1}{5}$ of a question each.

It is worth pausing a moment to appreciate the depth and beauty of Shannon's measure of information. It gives us an inescapable lower bound on the number of bits required to reconstruct data. In the roulette case this is about .2 bits. There is no way to do better, regardless of how hard we try or how clever we think we are. However, Shannon's measure *does not tell us how to achieve this minimum representation.* We know there is a strategy that achieves .2 bits per won/lost result, but we do not know what this strategy is. At first blush you might think this knowledge is not particularly helpful, but on the contrary it is enormously useful. It tells us how good it is possible to be, and when to quit trying to be more clever.

Let us return to the roulette example and see how we might encode the won/lost results to store them in a computer. Obviously we could begin by representing a win by a 1 and a loss by a 0. That requires 1 bit per result; we know we can do much better. In this situation, as well as in virtually every other case, the key is to package larger and larger amounts of data into each of the yes/no

questions that we ask. We will go into this at greater length later in this chapter and see why larger packages are more effective. For now, in this example we see that we cannot afford as much as one question per result, so we are forced to package more results into our questions. Suppose, then, that we adopt the ad hoc strategy of asking about the next 8 results as a block. Our first question is whether they are all losses. There is a good chance the answer will be yes, and we will have resolved the next 8 outcomes with just 1 bit. If the answer is no, we will have to get more specific. Suppose in that case that we straightforwardly ask about each of the next 8 results individually. The coding scheme is then as follows:

> 0 means the next 8 results are $L\ L\ L\ L\ L\ L\ L\ L$;
>
> 1$XXXXXXXX$ means the 8 X bits specify the actual won/lost pattern (for example, 100010000 means the next 8 results are $L\ L\ L\ W\ L\ L\ L\ L$) .

How well does our ad hoc scheme do? The probability that the next 8 results are all losses is $(\frac{31}{32})^8 = .776$. The average number of bits required to specify the next 8 results is $1\times.776 + 8\times(1-.776) = 2.568$. The average per result is thus $\frac{2.568}{8} = .321$. This is much better than 1 bit per result, but still somewhat more than the .2 bits promised by Shannon. So we can try to be more clever. For example, if we find that there is at least one win in the next 8 through a "no" answer to our first question, we could bank on there being probably only one win. Our next question could be whether or not there is only a single win. Probably the answer will be yes. Having established that there is only one win, we could use the divide-by-two strategy to pinpoint its location with the next three questions.

In the improbable case that there is more than one win, we could again revert to the brute force approach of asking about them all. I will spare you the details, but the average number of bits required per result using this strategy is .252. This is pretty close to the best we can do as promised by Shannon. We can quit trying. In fact, as we shall see later, the closer we try to get to the limit, the harder it becomes. The people who do not know about information theory will go on trying all kinds of devilish schemes to use even fewer bits. While this might seem farfetched, I can assure you that I hear all the time about such attempts. It is even somewhat fruitless to deter such people. Like the inventors of perpetual-

motion machines, they are determined. They do not want to know about Shannon or information theory. They believe that there is some magical encoding scheme that reduces the storage requirement indefinitely.

In later chapters we will be concerned from time to time with the information content of practical sources of information—English text, speech, and pictures. Unlike the simple examples we have considered so far, virtually all real sources are characterized by the fact that successive results (for example, characters, words, or picture elements) are not independent. We are not in a state of complete ignorance about the forthcoming result, but we have certain expectations that help us in our considerations. The fundamental uncertainty in forthcoming outcomes is reduced by knowledge of the past.

As a simple example, suppose that we want to encode weather data in the form of sunny *(S)* or rain *(R)*. The weather business these days is highly computerized, and predictions are fairly accurate, but some years ago there was a widely held opinion that a good forecast could be made by simply predicting that the weather would be the same as the previous day. Obviously, there is some correlation from day to day. Let us take this to the extreme and model the situation as if the weather depends statistically only upon the previous day. (Which is known in probability theory as a first-order Markov process.) Specifically, suppose that if it is sunny today, there is a probability of .8 that it will also be sunny tomorrow, and a corresponding probability of .2 that it will become rainy. Similarly, if it is rainy today there is a probability of .8 that it will also be rainy tomorrow, and a probability of .2 that it will turn sunny. A sequence of results might be as follows:

S S S S S R R R R S S R R R R R R .

Overall it is equally likely on any given day that it will be rainy or sunny. However, each day we have a definite bias. The uncertainty is diminished by the knowledge of the previous day's weather. We should not have to ask a full question to determine each letter in this sequence. As a matter of fact, if we merely rewrite this sequence in terms of whether or not the weather changed, using *C* for *changed* and *U* for *unchanged,* we get the sequence

U U U U C U U U C U C U U U U U ,

which is exactly of the form that we considered in the roulette example. The successive symbols are independent, but *U* is much more likely than *C*. The information conveyed by a symbol is

$$\textit{information} = -.8\times\log(.8) - .2\times\log(.2) = .722 \textit{ bits.}$$

We could come up with coding methods that packaged a number of symbols together and took advantage of the prevalence of *U*s, but as you can see there is not much to be gained. In the next chapter, when we consider text, we will describe an optimum method of encoding situations of this sort, called the Huffman code.

Often the interpretation of information from a sequence of symbols is philosophically difficult, for we do not always know the mechanisms of generation or the underlying probability distributions. Suppose that you are being given a sequence of binary digits:

1 0 0 1 0 1 1 0 0 0 0 1 1 0 1 _____ .

How much information is contained in succeeding digits? You might feel that you really have no indication whatsoever about whether any digit will be a 1 or 0. Perhaps someone is flipping a coin behind a curtain and calling one or zero according to heads or tails. You need to ask one question to discover each digit. "Is it a one?" you ask. For each succeeding digit you ask the same question. How could you possibly get by on fewer questions?

Another observer takes a different tack. I cannot see beyond the curtain, he thinks, but I suspect that there is a pattern behind these digits. For a while he is forced to ask the usual one question per digit. Meanwhile he analyzes the statistical properties of the sequence he is uncovering. Suppose, for example, that he discovers that there are significantly more 0s than 1s. In order to take advantage of this tendency in his questions he blocks his questioning to encompass more than one digit at a time. "Are the next three digits all zero?" he asks. If the answer is "no," then he tries the three sequences containing two 0s. If the tendency towards 0 continues he may well average fewer than three questions per block of three digits. He declares that the information content of this sequence is less than one bit per digit.

Still another observer might know something that we do not. Perhaps he can see behind the curtain and is able to ascertain that the sequence is being generated by a computer program. Instead

of asking about the output digits, he asks questions about the program that generates them. After a while he reconstructs the program himself and no longer has any need to ask further questions in order to predict the output digits. He runs the same program himself to reconstruct any desired length of output sequence. As the sequence length becomes very long, the average number of questions per digit approaches zero. In the limit he claims that the average information measure per digit is zero.

In many practical situations we are like the person confronted with the mysterious sequence being generated behind the curtain. Getting at the true statistics or the mechanism of generation is at least very difficult, and is perhaps even impossible. Almost no sequences are truly random and perfectly composed of unexpected events. Even the most innocent sources contain correlations, or redundancies, that lower their true information content. As a diversionary example of this truism, consider the next section.

SHANNON AND THE MIND-READING MACHINE—A DIGRESSION

We shall return to our explanation of information theory momentarily. In the meantime I would like to relate a story that will serve as a prelude for some of the issues that will interweave the remainder of this book, such as randomness and human behavior, as well as introduce a human element into what must ultimately become an abstract argument. Hopefully you will later feel some of the genius of Shannon's insight. Often when we are touched by genius, when we ourselves understand the vision unveiled thereby, we wonder about the person and the environment from which the original inspiration emerged. The story of the mind-reading machine, while by no means Shannon's alone, gives a hint of the people and the environment in mathematical research at Bell Labs in the early 1950s.

Shannon was an unusual combination of mathematician and tinkerer. Even today, in relative seclusion in a Boston home clut-

tered with gadgetry, robots, and toys, he occupies himself with the construction of juggling machines and the mathematical analysis of a stock portfolio that has grown to immense wealth. He has always been compelled by intellectual puzzles, yet at the same time his unquiet hands are those of a gifted craftsman. Today the problems he solves are diversions of his own choosing, left in drawers and unwritten thoughts. His ingenious toys never see the possibility of being taken to market. He appears rarely in public, and when he does it is usually only to claim the prizes that accumulate from the retrospective appreciation of the work of his youth.

In the early 1950s Shannon had completed his treatise on information theory and had won the freedom to indulge his whims for the construction of amusing and intelligent machines. At Bell Laboratories in Murray Hill, New Jersey, he built small artificial intelligence machines from relays and Erector sets. One was called Throwback, a calculator that operated in Roman numerals (evidence of a certain sense of humor!), and another was a machine that played a board game called Hex. In a day in which graphic displays were unknown, he built a machine that produced designs for logic circuits by dropping cards out of a slot to form a pictorial mosaic of the design. Perhaps his best known machine was Theseus, a maze-solving mouse. (In mythology Theseus slew the Minotaur, and by following the thread he had unwound on entering the maze where the Minotaur lived, he was able to find his way back out.) Theseus would be placed in a movable maze of metal partitions. It wandered about erratically until it reached the goal, but after that it was able to find the most direct solution and to memorize it for future attempts. The relay logic was hidden underneath the board, where Theseus himself was propelled by magnetic equipment. After Theseus, mechanical mice went out of style for 25 years, until microprocessors and a micromouse contest of the IEEE (Institute of Electrical and Electronic Engineers) breathed life into Theseus's modern progeny.

The mind-reading machine was the invention of Dave Hagelbarger, a kindred tinkerer at Bell Labs at that time. Hagelbarger lacked Shannon's mathematical sophistication, but like Shannon he brought ingenuity, mechanical visualization, and utter fascination to bear on the nearest interesting problem. Hagelbarger thought of Shannon as "a bright guy," but not all that unusual. Of course, Shannon would sometimes ride his unicycle down the hallway while juggling balls, and there were those occasional "clop clops"

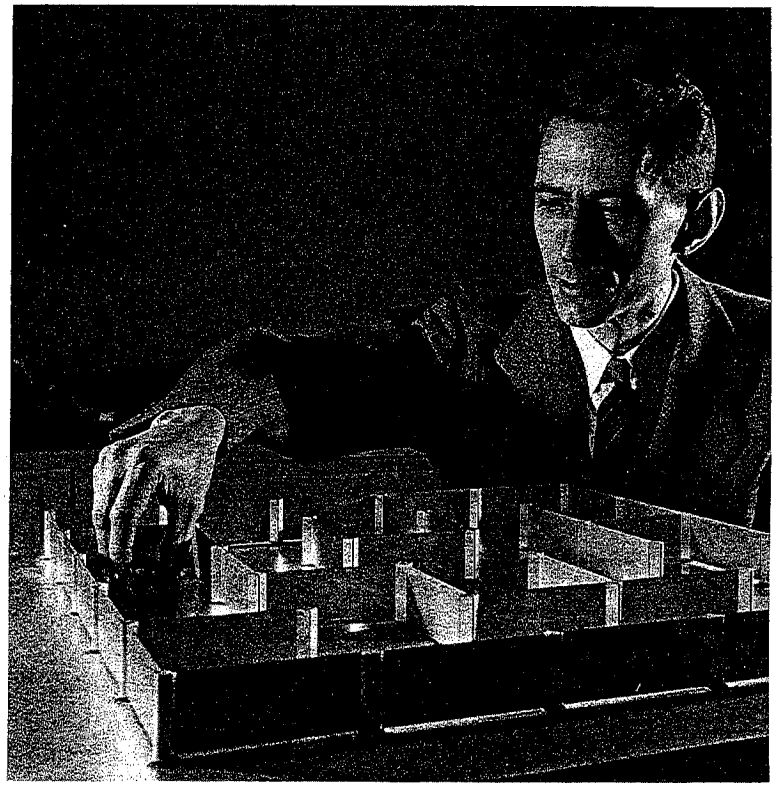


FIGURE 2
Shannon and Theseus, the maze-solving mouse

as Shannon went by on his pogo stick—hardly the image of today's corporate research! Now researchers in artificial intelligence seldom stray far from the screen of their computer terminals, and pogo sticks have disappeared from the corporate vocabulary. Perhaps a certain flair and individuality have been lost to the mounting sophistication and complexity of technological America.

The happenstance that brought about Hagelbarger's invention is probably a typical story. Three events formed a collusion. At that time Hagelbarger was gainfully employed in vacuum tube research, with nary a thought toward such an outlandish machine as what was to come. But over a fateful weekend an assistant accidentally left a plastic glove in the oven containing Hagelbarger's precious as-

sembly. The glove baked at 400 degrees centigrade for the entire weekend. On Monday morning Hagelbarger was greeted by a "gloppy mess," a vacuum system that had to be entirely rebuilt, and the ruination of months of effort. In complete shock, and with little else to do, he went to wander aimlessly in the library, freed from all that "gainful employment" of vacuum tube research.

The second event was Hagelbarger's discovery in the library of a fascinating book on telephone switching. Only recently had the art of telephone circuit switching been studied with the science of logic and Boolean algebra. (In fact, Shannon had discovered this link in a brilliant master's thesis at MIT.) Hagelbarger was completely captivated with the possibilities of design for "thinking machines" implemented with electrical relays. For days he was engrossed in reading the book on switching, and before long he made a connection with the third event, which was the mention by an associate of a science fiction story by J. J. Coupling in which a machine was able to create music in the style of various known composers by following their statistical patterns of musical notes. (J. J. Coupling was the pseudonym of another famous Bell Labs scientist, J. R. Pierce, who now four decades later spends his full time in the study of computer-composed music at Stanford.)

Hagelbarger now conceived the idea of a "mind-reading" machine that would try to beat a person in the game of penny matching by anticipating whether the person would declare a "head" or a "tail" at a given point in the game. Until now, you the reader have probably anticipated that this so-called "mind-reading" machine would be some kind of hoax. It was not. Hagelbarger built his machine, and over the course of 9,795 plays against human opponents the machine correctly anticipated the call of "head" or "tail" 5,218 times—a success rate that by chance alone would happen with probability less than one in 10 billion.

How is it possible for a machine to guess whether we will call "head" or "tail"? The explanation lies in our inability to generate a truly random sequence of calls. In fact, random numbers are quite hard to generate even in computers, and a good deal of research has gone into the development of various existing algorithms for random number generation. As we make calls to the machine we think to ourselves, well I chose a head twice in a row, now it is time for a tail. This is hardly a way to get at random calls, yet it is almost a universal approach. It results in the common fault that we choose too few long runs in our sequences—we change too often. For

example, if you were asked to call 16 head or tail choices, you would be unlikely to include a run of four consecutive heads or tails. Yet the actual probability in truly random throws of such a streak is about two-thirds. Because our "random" choices are innately patterned, Hagelbarger's machine could in effect read our minds by anticipating choices through a simple pattern-sensing algorithm.

The mind-reading machine had 8 internal states (a 3-bit memory) which it used to remember a limited amount of past data. These states corresponded to whether or not the machine had won the most recent match attempt, whether or not it had won the match the time before that, and finally whether or not it had played "same" or "different" the last time (that is, whether it changed from head to tail or vice versa). For example, it would know from recent play if it had a winning percentage on the next match after being in the "won-lost-same" state previously, and whether or not its next play should be "same" or "different." If the machine had a good winning percentage in the current state, it would decisively call its guess from its memory of similar wins. If the winning percentage was only fair, it would use its memory only 3 times in 4, interspersing random guesses one-quarter of the time. If the machine had been losing in a given state, it would guess completely at random in the subsequent match attempt. (The random guesses were obtained by the phase of a motor at the time the human guess button was pushed.)

On reflection, the machine's strategy is well matched to our own process of producing guesses. Our short term memory only goes back a few guesses, and the kind of considerations we take before guessing are similar—whether or not we have been winning, and whether or not it is time to change. The machine just does a better job in the bookkeeping. When the machine was first built, people quickly became bored with the simple penny-matching game, and Hagelbarger had a hard time collecting enough data. The addition of 2 rows of 25 lights across the top of the machine changed the game from boring to addictive. When the human won, the next green light in its row would be illuminated; if the machine won, it would be the next red light. Now most people played until one or the other row of lights was exhausted, indicating a "final triumph" for either the human or the machine.

If the machine's human opponents had analyzed its strategy, they could have beaten the machine 60 percent of the time. However, all they could see was that the machine kept winning more than its fair share of matches. One player got hooked to the extent that he spent his lunch hour at the machine every day for weeks. Eventually he began to win by unconsciously adopting a winning strategy. Another player was convinced that the way to win was to play randomly. His technique was to ask himself what he believed were randomly inspired "yes" or "no" questions to determine his calls. "Did I put on a red tie this morning?" he would ask himself. He lost consistently, so perhaps the questions were not so random after all!

News of the omniscient machine spread, and soon the director of research ordered the machine brought before him for a personal demonstration. Surprisingly, he won convincingly. Hagelbarger was at a loss to understand why the machine had crumbled so easily before authority. Since the director's calls had been recorded, Hagelbarger played back the exact same sequence of calls into the machine back in his lab. This time the machine won easily. The technological reasons for its previous failure remain unknown, but no scientist or engineer will fail to recognize the well-known syndrome of an experiment that refuses to work in the face of upper management.

Naturally Shannon had been following the performances of the mind-reading machine. At that time he had been intrigued with the workings of the human mind. Hagelbarger recalls that he and Shannon had bought an electroencephalograph machine to look at brain waveforms. Shannon had it hooked up to his own head, and Hagelbarger studied the waveforms. Apparently they looked just like anyone else's. (Scratch one theory of genius!) Someone had the inspiration to hook it up to the mind-reading machine. The EEG of the mind-reading machine looked just like Shannon's! This coincidence frightened people until it was realized that the motor that generated the random choices had a vibrational mode at about the same frequency as the alpha rhythm of the human brain.

Shannon decided to build his own version of the mind-reading machine—one which was smaller and simpler. His machine used the same 8 states as Hagelbarger's, but stored less information for each state and employed just two modes for its guesses, completely deterministic and completely random. Shannon used his usual flair for artistry in the method of displaying results of the matches. Two tracks with steel balls in glass tubes indicated machine wins and human wins by the balls popping out from the stacks.

Now the two mind-reading machines were pitted against each other in a celebrated duel. It was necessary to construct a third machine to act as a referee, and to dole out random numbers as required by either contender (since there was no human to push a button). For days the three machines huddled in a laboratory. Hagelbarger observed that if you told people that you were conducting an experiment in probability, no one was interested. But if you gave people the idea that two thinking machines were dueling to the death, then everyone was excited. In the end Shannon's machine won by a slight margin, but no one knows whether this was due to a happenstance trend in the random numbers or to a definitive edge in the algorithm that Shannon employed.

There is an interesting postscript to the experiment. One of the referees for Hagelbarger's paper describing his machine was an IBM scientist with access to a "real" computer, which he programmed to emulate a mind-reading machine with many, many states and total capture of all data. Curiously, it never worked well at all. Apparently the guesses of humans have little correlation over periods longer than that used by Hagelbarger, and in the short games typically played it is hard to get a meaningful amount of data about longer statistical trends.

ENTROPY AND INFORMATION

Shannon assumes a model for the generation of information in which a sequence of discrete symbols, representing messages, is chosen randomly from a finite set of possible symbols. We might imagine the telegrapher of Shannon's day being handed a sequence of English letters. His job is to send the letters; he does not reflect on what they mean. He comes to believe that they are being selected at random through some stochastic process that he is not allowed to know. As the days go by he begins to notice the statistical properties of the letters. For example, he notices that *e* occurs more frequently than any other letter, and that *t* is the next most likely. He observes that the letter *t* is often followed by *h,* and that *q* is almost always followed by *u.* He begins to realize that the block of

symbols *the* occurs quite frequently. All this kind of knowledge is perfectly compatible with the notion that the symbols are being produced randomly according to some statistical distributions. The telegrapher does not, however, read the messages. He attaches no meaning to the stream of nonsense symbols that he encodes with his rapidly clicking key.

The basing of his theory upon a probabilistic model was one of Shannon's important innovations. Twenty years earlier Hartley had observed, as we did previously, that information should be proportional to the logarithm of the number of possible symbols, but he did not make the connection to random selection and the mathematics of probability theory. There are two reasons why a probabilistic model of the world is often used in physical sciences. First, it acknowledges our ignorance of the true origins of the quantities involved and the minutia of their interrelationships. In effect we say that we do not understand all these details, so we pretend that the outcomes that we see—for example, the symbols that designate messages—are random. The second reason, however, is even more important. Probability theory is a powerful tool for analysis. Often when we are unable to calculate even one particular special case of known events, we find that we are able to calculate the *average* over many unknown, but random, events. So it was in information theory. Shannon's remarkable results are due in large part to the assumption of a random model of information generation.

Let me recount the measure for information that we described earlier. Suppose that we have a source of information that outputs a sequence of symbols, each of which is chosen from a set of n possible symbols. Then we found that the information per symbol is given by the expression:

$$H = -\sum_{i=1}^{n} p_i \log(p_i)$$

where the p_i are the probabilities of the n possible symbols.

Shannon called this uncertainty that measures information *entropy,* after a mathematically similar quantity in classical statistical mechanics. In statistical mechanics the entropy is a measure of the disorder, or uncertainty, in a system. Information is similar in that it also may be regarded in terms of the fundamental uncertainty. The entropy represents the state of our uncertainty as to the actual symbol values, which is equal to the information that needs to be

provided to reconstruct them. Some of the mystique of information theory is due to this coupling with the entropy of statistical mechanics. The second law of thermodynamics, which says that entropy is always increasing, gives an almost metaphysical quality to entropy. That the entropy in an isolated system cannot decrease would mean by analogy that information, representing order and organization (the resolution of uncertainty), could only in some large sense be created locally at the expense of increased disorder elsewhere. Such philosophical arguments might be made appropriately late at night over a bottle of wine, but they have had really nothing to do with the evolution of information theory. It is probable that any analysis of order and disorder would result in similar equations. The extrapolations from the world of statistical thermodynamics, tempting as they are, have not yielded insight into the world of information and communications.

Examination of the equation shows that it is at a maximum when all the symbols are equally likely, that is, when $p_i = \frac{1}{n}$. This is the state of least prior knowledge. If some symbols are more likely than others, then naturally the uncertainty is less. Similarly, the sum of individual information events in this equation is made simple by the assumption that successive symbols are independent. If they are not, then the entropy of the source is again less, since knowledge of past symbols implies information about symbols yet to come. The entropy under these circumstances can be calculated by enumerating the possible states the source can attain and evaluating the conditional entropies associated with each state and the probabilities of transitions to other states. This can obviously be a matter of some bookkeeping, and need not concern us here.

Real life channels are invariably messy. Data sources like text, digitized speech, and pictures have complex dependencies that extend deep into the structure of the sequence of symbols. In most such cases the calculation of source entropy is really impossible. The most important example is the sequence of the symbols we call "letters" in the English language. How many bits does it take to specify each letter on average? This is a question we shall discuss in the next chapter. Along the way I hope you will be fascinated, as I have been, with the beautiful complexities and redundancies of our language. For the present, let me give a little appetizer in the form of an illustration of the manifest redundancy in English text.

How many yes/no questions does it take on average to specify a letter in English text? Since there are 26 possible letters, not

counting spaces and punctuation, we might believe that it would take about 5 questions per letter. However, I remember from my childhood the fascination I had with the radio show "Twenty Questions." Perhaps you know of it or have played it yourself. The host would begin with, "I am thinking of something which is mineral (or animal or vegetable)." If the panel could not find the secret, mineral thing within 20 "yes" or "no" questions, then the contributor would receive some prize. Surprisingly, to me, the panel nearly always named the thing within the 20 questions.

If the game of 20 questions really works, then we can also say that nearly any noun in the popular knowledge domain can be represented by about 20 bits. If you feel this number is evidently sufficient, then think for a moment that the average length of words in this book is 6 letters. If each word can indeed be represented by 20 bits, then this means each letter can be encoded in approximately 3.3 bits. However, the way my computer works is that each key I strike on my word processor is encoded into an 8-bit byte and stored in the computer memory. Such a waste. Obviously, my computer does not know about information theory. We will return to these interesting philosophical questions in the next chapter.

In more recent times the radio shows have gone, but our interest in language games remains a constant. The television show "Wheel of Fortune" is in its way also a search for the entropy of English. A secret phrase is revealed only as the contestant is able to guess the letters it contains. In the process the contestant is given some "free" information in terms of the locations of the letters guessed within the phrase. In order to convert the average number of guesses required to the entropy of text, we would have to add the questions necessary to pinpoint the letter locations. With this minor, and easily calculated, modification, we could get some idea of the information content of English text through the number of guesses used week after week on "Wheel of Fortune." Of course, the contestants could care less about this interpretation.

While it is true that it is essentially impossible to calculate the entropy of a real source like English letters directly from the mathematics of information theory, the principles nonetheless serve for inspiration and insight. The coding of a real source usually devolves from artful, if ad hoc, methodology. In succeeding chapters we will discuss some of the coding methods for text, speech, and pictures that are used to squeeze out unwanted redundancy and lessen storage and transmission requirements. Information theory sets out this

beautiful challenge. Perhaps it is just as well that it neglects to tell us how to achieve this minimum representation. Otherwise it would be too easy.

ENTROPY AND THE INFORMATION PYRAMID

This seems like a good time to step back from Shannon's microscopic view of information to put in perspective the larger context of information in the sociological realm as discussed in the previous chapter. In that chapter we presented an information pyramid—data, information, knowledge, and wisdom. The difference between the layers of the pyramid was one of increasing organization as we approached the top, which was wisdom. We pictured the "information" layer itself as being the condensation through organization of the sludge we called data lying underneath.

In Shannon's theory we find the concept of information as the resolution of uncertainty. The measure of the uncertainty is called entropy, which is directly related to the degree of randomness or permissible disorder. Increasing entropy means decreasing order and structure. Order and structure in Shannon's framework represent redundancy, which diminishes the intrinsic uncertainty that is resolved by the receipt of information. Consequently, the more structure in the underlying process, the less information is conveyed by a specific observation of the process. On the other hand, the more freedom or allowable disorder in the underlying process, the more information can be conveyed.

In either the technological or the sociological sense information is a question of order and disorder. Shannon's theory is a technological or mechanical concept applicable on the bottom (data) level of our pyramid. It tells us about the size of containers we will need for our sludge; it gets to the essence of the sludge, the minimum storage that will contain the data. The more disorder that is permitted by the structure of the sludge, the more information the sludge can yield. For example, a demented monkey banging randomly on a typewriter generates more Shannon-type information per character than I am currently producing here on my terminal. (Actually, the monkey need not be demented, but more on the monkey in the next chapter.) He does not have to spell correctly, follow the rules

of grammar, or make semantic sense. I envy him his freedom. It would take more bits per character to store the monkey's output than to store mine, which is so much more constrained. For the same reason, you would find it much more difficult to memorize the monkey's characters than mine. It is not just a question of storing information in a machine; it is us too. We also are sludge containers.

Moving away from the sludge level to the personal and sociological concepts of information, we get a different perspective on information. The first key human job—and no machine can do it as effectively—is to distill the sludge, so as to boil off the unneeded or unwanted data. At the Shannon level the characters typed by the demented monkey were treated the same as the characters of this book. (Although sometimes I wonder. If I could actually find a demented monkey, I might be able to verify this.) There is no concept of value or relevancy. The information content is the measure of what is needed for perfect reconstruction. But in the real world we must exercise great art and skill in filtering or selecting among the possible information sources to screen for value, relevancy, and truth. Reject the monkey. Keep this, I hope.

After selection, to move the information to higher levels of value we need to integrate the information with our existing knowledge. We need to create interlocking links with the storage and filing structures that organize our knowledge, and we need to find ways to make the new information memorable. These actions will inevitably add redundancy to the newly acquired information, but this is redundancy of our own choosing, designed to protect the integrity of the new fragile bits. All of information processing can be seen in similar light. First we squeeze out undesirable redundancy, then we add desirable redundancy, and so forth. In the next section we will see a mechanical reason for adding redundancy back into data when we begin to worry about the vagaries and risks to information in communications.

COMMUNICATION CHANNELS AND CAPACITY

Let us turn our attention from data sources to the communication channels, which transport symbols. For our purposes the channel

itself is a mysterious black box (these things get around a lot) that accepts a sequence of symbols at its input and delivers another, hopefully similar, sequence of symbols at its output. Think of a telephone connection sending data between computers or of a computer sending data to a disk drive, for example. The input and output of the channel are often geographically distant; they may be literally worlds apart. But all the complexity in the transmission system that transports the symbols is subsumed into a statistical knowledge of the transition relationships between input and output symbols. Sometimes it makes an error, mostly it does not. That kind of thing.

Shannon was remarkably prescient in thinking of and modeling the communications channel as a statistical passage of discrete symbols. In the late 1940s digital transmission was almost completely unknown. Pulse code modulation had been invented in 1939 by Reeves (of ITT), but it was not until 1961 that the first digital transmission system was deployed by the Bell System. Even in the 1950s very few communications engineers understood the philosophy of digital transmission. The crux of Alexander Graham Bell's invention of the telephone some 75 years earlier had been the use of *analog* representation, where the transmitted voltage was proportional to the sound pressure at each instant.

Today essentially all newly installed communications channels are said to be digital. I use the phase "said to be" because clearly at some point in the system analog voltage or lightwave pulses are used to convey the 1s and 0s that are decided at some distant point. Nevertheless, nearly all of us think of the channel as digital. We shove bits in one end of the connection, and they pour out at the other end. In between who knows what happens? Moreover, not many people care. The modern attitude is that the overwhelming majority of systems designers of the information age will think of communications channels as transporting bits, which are occasionally corrupted in some unfortunate way. The details of the actual transmission are relegated to a small subset of the technological world, whom we might think of as the "keepers of the channel"—engineers who design modems, microwave and satellite systems, etc. I do not mean to denigrate their role, since I and many of my friends make our living inside that black box, but in the deepening complexity of the information age it is often necessary to practice the hiding of lower-level details in a hierarchical

fashion. Thus to most of us the communications channel, just as Shannon foretold, is merely a mover of symbols, and since we live in a binary world, those symbols are specifically binary digits, or what we call bits.

Inside the communications channel terrible things happen to our precious bits. The indignities they endure are remarkable. The bits are converted to pulses which are variously subjected to distortions, random voltage fluctuations known as electrical noise, electromagnetic interference, and assorted catastrophic events. Every now and then a little box in a manhole called a repeater decides whether or not the pulse it is looking at more resembles a 1 or a 0. Having made its irrevocable decision, the repeater generates a clean new pulse of the decided variety, which now travels to the next repeater. Such is the beauty of digital transmission; distortion is not allowed to accumulate. As long as the repeater decisions are correct, a series of virgin pulses is regenerated. Nonetheless, life is full of imperfections, and errors are inevitable.

In Shannon's time, electrical noise, caused in all electrical circuits by random, thermal motion, was considered the sole cause of errors and to be the limitation to transmission speeds. Today, while thermal noise may be the ultimate limitation, there are so many uncharted causes of transmission errors that few researchers seek descriptive mechanisms for error generation. Murphy's law overwhelms everything—errors happen, who knows why? But granted that errors are inevitable, what is their effect on the transmission of information? This is where Shannon's greatest contribution lay.

Suppose that we transmit a sequence of bits over the channel, and errors occur where I mark "X".

```
10110100011011100101111001011000
    X          X     X
```

Therefore we receive the sequence:

10110000011011100111111001111000.

This looks like a perfectly legitimate sequence to us; we have no way of knowing that it contains erroneous data. In order to reconstruct the original data we would have to know the error locations. The error locations could be represented by a sequence

of bits with 0 for "no error" and 1 for "error," giving the following description:

0000010000000000001000000100000.

If we only knew this sequence, we would know which bits had to be changed (inverted), and we could fix the received data. This error sequence is the unknown, unprovided information. It is the result of the uncertainty added during transit by the communications channel. How much information does it take to learn this sequence and fix our data? Well, we just finished deriving the quantity of information necessary to describe a sequence exactly like this; it was what we called entropy, and we have a general equation to quantify it. In the case of independent, binary symbols, as above, the general equation for entropy simplifies to

$$H = -(p \log p + q \log q)$$

where p is the probability of the symbol 1, representing an error, and $q = 1 - p$ is the probability of the symbol 0, representing no error.

Somehow the received binary digits, as they are, are "missing" H bits of information. Before transmission each digit was worth one whole bit of information, but the channel subtracts H bits per digit of information because of the added uncertainty. Another way of looking at the transmission problem is to recall that we thought of the entropy H as *uncertainty.* In transmission what we are looking for is the opposite of uncertainty; it is what is left after removing the uncertainty, and it represents *certainty.* That certainty is now worth only $1 - H$ bits.

Shannon defined the *capacity* of a channel as its intrinsic ability to convey information. For the binary channels we have been discussing he proved what we might now suspect—that the capacity is simply $1 - H$, or

$$C = 1 + p \log p + (1 - p) \log (1 - p) \textit{ bits per binary digit.}$$

(Again, I hate to add caveats, but this simple equation assumes that the errors are independent events. In real life errors almost always cluster together as if they wanted company. This dependency decreases the entropy and thus increases C.) Now perhaps with this

background the reader will not be surprised by Shannon's fundamental theorem, which is

> *A channel with capacity C* is capable, with suitable coding, of transmitting at any rate less than C bits per symbol with vanishingly small probability of error. For rates greater than C the probability of error cannot be made arbitrarily small.

Let me rephrase the meaning of this theorem. The problem is that the raw symbols (the 1s and 0s) received at the channel output have an uncertainty associated with them; in effect they are not worth a full "bit" of true information. Some of the information that we thought we could deliver at the receiver will have to be devoted to another cause, that of uncovering the locations of the errors. Shannon's fundamental theorem assures us that there is a way of using some of the received bits to locate and correct the errors, and that if the fraction of bits that we sacrifice for this use is larger than the channel entropy H, then it is possible to find and correct virtually all of the errors. If we devote a fraction of bits smaller than H, then Shannon tells us that regardless of how clever we may think we are, it cannot be done.

Actually, Shannon's theorem does not say that we can eliminate *all* errors, but only that we can make the residual error rate vanishingly small. By "vanishingly small" we mean that if we work at it very hard we can make the error rate as small as we like. Although I would like for the sake of simplicity to say that *errorless* transmission is possible at rates less than capacity, the correct mathematical interpretation is that errors will still occur, but the fraction of errors can be made arbitrarily small.

I wish I had found a way to tell those readers who have not previously studied Shannon's work how magnificent and surprising this seemingly simple result is. Unfortunately, it takes a certain depth of insight that I feel I have failed to convey to the reader in order to realize the profound implications of the concept of channel capacity and its promise of errorless transmission in spite of channel errors. In his book *Symbols, Signals, and Noise,* John R. Pierce (a contemporary of Shannon at Bell Labs) observed that lay people thought this an unsurprising result, while engineers and scientists were shocked by it. Its impact on the communications engineer—once it was understood, which took some years—cannot be underestimated. You might reason that if a channel has a capacity C bits

per symbol, then of course you should be able to send data at rates up to *C*. Obvious. But the catch is that the error rate must be made arbitrarily small at the same time that the communication rate is maintained at *C*. That this was possible was a revelation to the engineers of the early 1950s.

Here is how the engineers of those days reasoned. They felt that it was obvious that the error rate in a noisy channel could be made arbitrarily small. For instance, if you wanted to send the data 101 over a channel that occasionally made errors, you could simply repeat each bit three times and make a majority decision. Go for the best of three, you think. So you send 111000111. But there is still some residual probability of making an error. It might happen that two of the three bits are in error. Simple, you think, I will just go for best of five. So you send 111110000011111. Let the channel try to corrupt that, you exult. However, there is still some small probability of getting three errors in a block of five, and you must assure a "vanishingly small" error rate, so maybe it should be best of seven, or best of nine, or whatever. Anyway, eventually by this strategy you will reach whatever is considered a "vanishingly small" error rate.

But look what is happening. Sure, the error rate becomes smaller and smaller as we go for best of seven and best of nine, etc. But at the same time the data rate is getting smaller and smaller. Only one bit in seven carries information, then one bit in nine, etc. By the time we reach a vanishingly small error rate, we will also have a vanishingly small information rate! The stunning import of Shannon's theorem is that we can have our cake and eat it too. The probability of residual errors in the data can be made as small as we like without suffering any diminution of the data rate, as long as the data rate itself is less than the channel capacity. There is *some* way to get virtually errorless transmission *at the same time* as we achieve data rate *C*. Truly amazing!

Reflect for a moment that entropy in a *source* is something that we seek. The entropy of the source measures its "richness" in information content. Uncertainty is good in the sense that the raw bits from the source then carry higher value, whereas if the entropy is relatively low the raw bits are highly redundant. On the other hand, entropy in a *channel* is bad. Here we look for as complete a certainty as we can obtain. Any "unknown-ness" about the channel will diminish its intrinsic information-carrying capacity. So the philosophical approach to communications is the following. First we

compress the data output from the source to remove redundancy, then we expand the compressed data to protect it against the vagaries of transmission through the channel. At the receiver the operations are reversed. This is shown diagrammatically in Figure 3. Communications engineers are always thinking of squeezing out the redundancy in the data at the source, and then of pouring redundancy back into the stream for protection at the channel. The

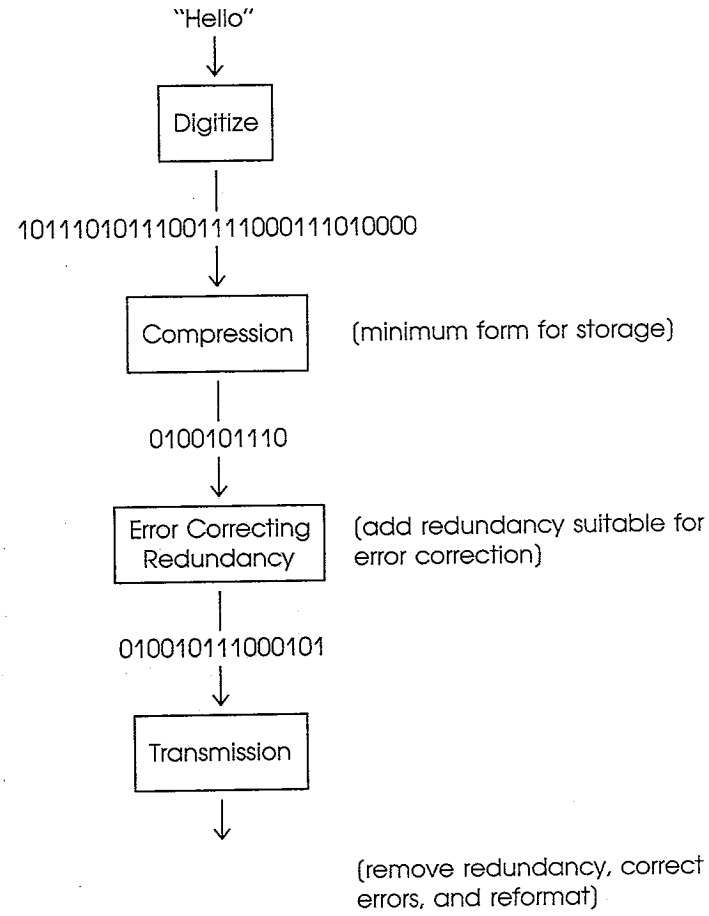


FIGURE 3
Transmitting source material through a channel

natural redundancy of the source bits is seldom suited to provide the specific protection we need at the channel.

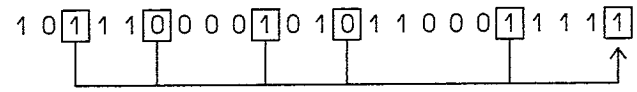
ERROR CORRECTION BY CODING

I often reflect on how marvelously we humans have adapted our natural communications methods to our environment. In our spoken language we too have to deal with a "noisy" channel that is uncertain in its correctness of transmission. For that reason we have evolved a vocabulary for transmission that includes redundancy that protects against channel uncertainties. Our speech is incredibly robust, and our detection mechanism unbelievably sophisticated. Consider the so-called "cocktail party" effect where we engage in conversation with someone in the midst of near bedlam. In spite of the blaring music and the presence of numerous, competing conversations all about, we are able to carry on a meaningful (perhaps!) conversation with the person on whom we focus our attention.

The central factor that builds robustness into language is the limitation of the vocabulary to a minute subset of the possible combinations of allowable letters. In English with 26 letters there are about 10 billion possible combinations of 7 letters which might be words. We get along on about 50,000 of these possible words. When we hear or read a word that is not one of those we recognize, we automatically search in our minds for possible words that are "close" to the garbled word we have heard or read. Communication codes do exactly the same thing with data words. However, we humans excel in even going much further than the electronic decoders; we are able to use the context and our entire experience and knowledge base to aid in our detection. There is redundancy woven deeply into our language. If we are sitting at the dinner table, and you hear me say, "Please pass the guther," you reach for the butter without conscious hesitation. That was an easy one, but think for a moment about how very, very good we are at such extraordinarily complex processing.

In contrast to the historically evolved redundancy in natural language, the computer data typically handled in communications channels either is, or appears to be, devoid of redundancy. If we have done a perfect job in source coding, the information rate of the stream will be one bit of information per symbol—every bit that we see really counts for a bit of true information. It is also possible that at the communication channel we may not be allowed to meddle with the bits we see. Like the uncaring telegrapher of Shannon's day, we may not be permitted to interpret meaning in the relentless flow of symbolic script. In either case every sequence of bits the channel handles is a legal vocabulary word. The block 10010111 makes as much sense as 00111010. Imagine trying to converse when every combination of letters represented an allowable word! Clearly, if the channel coding is to protect against errors, we must restrict the allowable data sequences to some subset of all those possible. Then, just as in human conversation, we check the received data words for validity. When we find an invalid word, we can search for the nearest legal word—nearest in the sense of being the most probable origin of the garbled word we have received. Alternatively, again analogous to human conversation, we could notify the transmitter that the word was garbled, and ask for a retransmission. ("Huh? Would you say that again?")

The error-correcting philosophy that comes from Shannon's information theory is the idea of devoting some of the bits to the purpose of locating and correcting possible errors. These so-called check bits add redundancy such that only certain combinations of data bits and check bits are legal data words, just like only certain combinations of letters produce words in English. Each check bit is in effect one of our yes/no questions, designed to ask a question that helps to identify the error sequence. For example, a question it might ask is whether or not there is an odd number of 1s in some predetermined subset of the bits, as indicated below.



If there is an odd number of 1s in these boxes, then this check bit will be 1, otherwise it will be 0.

If we devote a fraction H or more of the bits to the purpose of checking, then there is sufficient information carried by the check bits to balance out the entropy of the channel and, *on average,* to correct the errors in transmission. However, that "on average" qualifier is terribly important. We only get one crack at it. Is there or is there not enough information in the check bits to correct the errors in our particular data sequence? Shannon assures us that if we ask the right questions with our check bits, and take the right actions based on the answers we get, then it is possible to get an error rate as small as we wish as the data sequence becomes longer and longer, while keeping the same fraction of check bits.

The first error-correcting codes were discovered by Richard Hamming at about the same time that Shannon was conceiving information theory. In Hamming's code, the data is grouped into 7-bit blocks, composed of 4 information bits and 3 check bits. The check bits are determined before transmission by the oddness or evenness of different combinations of these 4 information bits. At the receiver, the check bit calculations are repeated to see if the received check bits agree with the calculations. If a check bit disagrees at the receiver, then there are one or more errors in the data bits for which it is responsible (or the error can be in the check bit itself). By looking at the patterns of discrepancies in the received check bits, we can calculate the most likely error pattern that would have led to this particular set of check bit discrepancies. We then correct this most likely error pattern. Most of the time we will have fixed the errors; some of the time a less likely pattern (containing more errors) will be the actual cause of the discrepancies, and we will unknowingly correct the wrong thing.

For Hamming's famous code a little analysis shows that we will always do the right thing as long as there is only one error in the block of 7 bits (the 4 information bits and the 3 check bits). If there is more than one error in the block of 7 bits, then we mess up. Thus we sacrifice $\frac{3}{7}$ of the bits to correct any one error in 7 bits. As a result, the error rate is reduced, but not indefinitely as Shannon promised.

There are many other more complicated coding algorithms that have been discovered in the last forty years. The subject is a highly mathematical one, based largely on the theory of modern algebra, which is an abstract generalization of the algebra with which most of us are familiar. There was a time when most of us thought that modern algebra was a beautiful, but useless, recreation for mathematicians. Then it was discovered that error-correcting codes could be synthesized, and their performance guaranteed, by basing the code construction on the principles of modern algebra. The only catch was that there was not much use for these sophisticated error-correcting codes either.

I can remember the first time that I read about Reed-Solomon codes. This sophisticated coding algorithm was a generalization into an even more abstract space of the already abstract codes that were then known. Uselessness piled onto uselessness, I thought. That was then. Today I use a Reed-Solomon code for my own enjoyment almost every day. Perhaps you do too; it is in every compact disk player. The compact disk playback system is like a communications channel. Errors can occur because of imperfections in the disk, or because of dirt, or maybe because you scratched it. Anyway, you might not have realized that virtually all of these errors are found and corrected because of the redundancy built into the recording format using a very powerful Reed-Solomon code. You can run a nail file across a compact disk, and you will not hear a single hiccup on playback. Every bit of the music will be exactly as it should be. I have not actually had the courage to try this, but the theory guarantees it. The coding system is capable of correcting as many as 8,232 consecutive errors. Think about that; it puts those little redundancy checks into perspective!

During the course of writing this book I have stored the growing file of characters from my word processor again and again on hard disks and floppy disks. Thousands of times the file has gone back and forth, from the memory chips of my computer to the disk and back again. Each time the integrity of the bits has been ensured by an error-detecting code. In this storage application the data is accepted only if all the check bits are correct, otherwise the whole operation is repeated. Moreover, I have transmitted the files of characters back and forth between locations around the world and my home and office. I have done this hundreds of times, with a file that is almost a megabyte in size. Each time the file was first compressed to remove needless redundancy by an algorithm that we will discuss in the next chapter, resulting in approximately a halving of its size. Then about a 1 percent redundancy was added to the file in the form of check bits computed according to a standard code used in communications protocols for information exchange. If any block of data (typically 1,024 bytes) failed its checks, it was retransmitted. I never worried. There was no chance that so much as a

comma would be lost. If you have ever written a large manuscript, you must know what these little characters meant to me, yet I entrusted them to codes. I never imagined that codes would be so important practically, but as you can see, they are.

MAKING AN EXAMPLE SMALL CODE

I have resisted so far the compulsion to describe in detail the construction of codes. This is, I keep telling myself, a "how come" book, rather than a "how to" book. When you lift the hood and look inside coding and decoding algorithms, you see a gleaming engine, with many moving parts. I could discuss how this lever moves that pushrod, which is attached to that wheel, and so forth. In fact, I am so proud of that polished engine that a little later I will succumb to the temptation and show you a gear or two! But too much of this and I know I would get lost in detail, and so perhaps would you. The principles that run the engine are some of the most compellingly beautiful mathematics that I know, but I fear that the detailed description would take us away from the foundations of information that form the theme of this book. Instead, I would like to take you through an exercise of making our own little code that uses no mathematics whatsoever. We will just pick code words at random, just as if we made up an English dictionary by drawing tiles from a Scrabble rack. However, even without mathematics I fear that the following sections are the most conceptually difficult in this book. Feel free to skip ahead. After all, that is what the information age is all about. And if you stay through and understand, then you will have reason to be proud of yourself.

In order to get at the general philosophy we will use a simple, concrete example. Suppose that the channel has a probability of error of .02, so that on the average 2 percent of the 1s come out as 0s, and vice versa. You have to suspend reality for a while because real communications channels are usually engineered for a probability of error of about one in a billion, but this channel is bad. When we calculate the channel capacity we find

$$C = 1 + .02\log.02 + .98\log.98 = .8585$$

We might notice in passing that even a small percentage of errors cuts down the channel capacity rather appreciably. According to Shannon's theorem, we ought to be able to use this channel for accurate transmission as long as only about 85 percent of the bits represent delivered information, with the other 15 percent of the bits providing redundancy and protection. Since we will not be greedy in this example (for good reason, as we will see), let us aim to get only half of the transmitted bits through correctly—half of the bits will represent information and half will be what we call check bits.

Actually, there is no need to think of particular bits being designated check bits, and other bits information bits, any more than we think of English words as being composed of "information" letters and "check" letters. The point is that we have a dictionary, and some sequences of letters are in it, while others are not. We can do the same thing with bits by selecting a dictionary of "approved" sequences, or what we will call code words. Then we make some kind of arbitrary assignment of information sequences into code words in the dictionary. As a specific example, suppose that we begin with very modest sizes of words in our vocabulary, using only 4-bit code words. Since the code rate is to be one-half, that means that every 2 bits that are input as information will dictate a particular 4-bit word from the dictionary to be transmitted. Since there are only four possible combinations of two bits (00, 01, 10, and 11), we need a dictionary that lists only four words. We could decide, for example, that when we get the input 00 we will send the code word 1011, and when we get input 01 we will send 0110, etc. There are 16 possible 4-bit combinations, but only 4 of these will represent legal code words. In this way the code words will be only a subset of all the possible combinations of channel bits, and we should be able to recognize when the words become garbled by channel errors. The price we pay, obviously, is that the effective information rate of the channel is cut by 50 percent.

The first problem we face is how to make up the dictionary. We need to designate 4 of the possible 16 combinations of 4 bits as code words. The number of ways that we could make this selection would be $16\times15\times14\times13$. That must seem like a lot of possibilities for such a simple problem (though indeed there are so many symmetries that far fewer possibilities are really significant), but in truth we have not touched the real difficulties yet. Surely, you imagine, there must be some optimum way to select the dictionary.

Some choices of code words must be better than others. As a rough guiding principle, it seems evident that we would like the code words to be as "different" as possible, so that they will be less likely to be confused by channel errors. For instance, we would prefer not to have both 1111 and 1110 in the code, since a single error in the last position would convert one code word to another, and we would have no way of knowing that the error had occurred. Indeed, an optimum way to select this particular dictionary is known, but only because the code is so small. For the moment we will just pretend that we do not know how to make an optimum code.

The first instinct of a mathematician facing the problem of determining how well a code can do would be to optimize the code dictionary, and then to find the performance of this best code. Lacking the ability to find an optimum code, most mathematicians would search for bounds on the performance of codes. In this vein Shannon found an ingenious lower bound through the observation that the best code must be at least as good as the average of codes picked randomly. The best must be at least as good as the average. That seems obvious enough, but it also sounds rather unhelpful. However, this so-called random coding bound turned out to be much more insightful than might have been anticipated. Let us follow this line of thought with our example. For the moment we do not know how to construct the best possible dictionary. Instead, we choose the code dictionary randomly by tossing a coin to write each bit in each code word it contains. I formed a number of such random codes on my computer. (I wonder what Shannon would have done with a computer like mine. Perhaps he would still be playing with it and information theory would never have happened!) One of the better codes can be found in the following table.

EXAMPLE RANDOM CODE: 4-BIT WORDS, .5 INFORMATION RATE

INFORMATION	CODE WORD
00	1001
01	0000
10	1111
11	1110

To see how such a code might be effective suppose that the information block to be transmitted is 01. In our randomly constructed dictionary we find the entry 0000 as the code word to be transmitted. Suppose further that during transmission the third bit is in error, so that we receive the illegal word 0010. Searching through an identical dictionary at the receiver, we compare 0010 with each of the possible legal entries. This sequence differs from the first code word (1001) in three positions. Thus if the first code word had actually been transmitted, it would have taken 3 errors to result in the sequence that we received. Similarly the "distance" in errors from the other 3 legal code words is, respectively, 1, 3, and 2. The closest allowable word is the second word 0000. That is, it would have taken the fewest errors to convert that word to the word we received. Such a "maximum likelihood" decoding procedure results in minimizing the probability of error for a given dictionary. In this instance the decoding will result in correcting the error in transmission.

It is simple enough to determine the performance of our sample code. There are 4 possible, equally likely, code words that can be transmitted. In the channel one of 16 possible error patterns will be added to the code word. (We use "added" as the logical exclusive, that is, a 1 becomes a 0 and vice versa.) These error patterns are not of equal likelihood. The most probable error pattern is 0000, indicating no errors. It has a probability of $.98^4 = .922$. Next most probable are the four patterns with a single error: 0001, 0010, 0100, and 1000, each of which has a probability of $(.98^3)(.02) = .019$. Similarly there are six double error patterns, 4 triple error patterns, and one pattern in which all 4 bits are in error.

To evaluate the probability of error for the coded system we take each of the 4 possible code words, add each of the 16 possible error patterns and determine, as we did above in one of the instances, whether or not decoding by finding the closest allowable word will result in the correct decision. (Often it will happen that a word to be decoded will lie at equal distance from a number of possible code words, and we make an arbitrary assignment.) The product of these possibilities means 64 situations must be evaluated, but that is trivial in this example. We find that 62 percent of the time that a single error occurs in transmission it is corrected by the decoding procedure, and that 8 percent of the time that 2 errors occur the decoding is correct. We are out of luck if 3 or 4 errors

occur in the block. Weighting these events by their probabilities of occurrence results in an overall probability of code-word error of .0304.

Summarizing for the moment, we took a raw channel with a probability of error of .02, and traded half of the transmitted bits in order to build redundancy into the stream. We did not know how to construct an optimum code dictionary so we made one up at random. At the receiver we used closest distance decoding, which we know is the best that can be done. The overall result of the experiment was that the code words have a probability of error of .0304 after decoding. Is this good? Well, not very. If we had sent 2-bit blocks without the 2 extra bits, the probability of error for the comparable blocks would have been $1 - .98^2 = .0396$, which is somewhat higher than our coded system. Thus the performance is indeed better, but admittedly not by a lot!

In a simple case such as 4-bit words it is easy enough to find the optimum dictionary with 4 entries. The code words are a little further apart from each other than the words we picked at random. In going through the exercise of finding the probability of error you would find that the optimum code corrects 75 percent of the single-error events, and none of the less probable error events. It has a probability of error of .0212. Still not great. What can we do?

TOWARD LONGER CODES

Shannon promised us we could reduce the probability of error indefinitely. We did everything we could with our little dictionary of 4-bit entries. The only thing left to us is to go to larger dictionaries. Suppose now that we consider 8-bit code words. Since the words will have to carry 4 bits of information, there will be $2^4 = 16$ code words, chosen from a possible set of $2^8 = 256$ sequences. The number of ways the dictionary could be constructed is $256\times255\times254\times \ldots \times250\times249$, which is obviously a huge number, but again we can choose the dictionary entries randomly and acknowledge that the optimum code must be at least as good as our random choice.

My computer spewed out a number of randomly generated codes and evaluated their performances. There was not a great deal of spread in the final numbers, with the best probability of error in this lot being .0251. This particular dictionary corrected 87% of the single-error patterns and about 27 percent of the double-error patterns. Here again the optimum code is relatively easy to find, and with a small amount of cleverness we can determine that it corrects all 8 of the single-error patterns and 25% of the double-error patterns. It has a probability of error of .0079. Now we seem to be getting somewhere. If we had merely transmitted 4-bit information blocks without the redundancy of the 8-bit dictionary, the probability of a block error would have been $1 - .98^4 = .0776$. At least now it seems that we are getting some payback for giving up half of our bits for protection. The clue seems to be to go to longer codes.

Flushed with success, we would like to move on to 16-bit dictionaries. However, there is a big, big problem developing; it is the tyranny of exponential blowup in the complexity. It is very important for us to understand this because it is central, not only to the specific coding problem that we are discussing, but to so many of the important problems of information handling and processing. As we make longer code words we seem to be getting better performance. From what we have seen in our example, the progress is slow, but encouraging. We will examine momentarily the philosophy of why longer code words are better. Meanwhile look at what is happening to the complexity of our system. When the code words were 4 bits long, there were only 4 entries in our dictionary, representing the 2^2 possible 2-bit blocks to be encoded. To evaluate the code we looked at 16 different error patterns in combination with these 4 words—a total of 64 possibilities. Easy. With 8-bit words there are 2^4, or 16, entries in the dictionary. Still easy. In order to evaluate the codes we combine these with 2^8 possible 8-bit error patterns, which results in 4,096 possibilities. Modest, but we raise our eyebrows. Now to do 16-bit code words there are 256 code words and 2^{16} possible error patterns, which makes 16,777,216 possibilities to evaluate. I am just not going to be able to rummage with this on my little computer, or even on most of its bigger brothers. And 16-bit codes are tiny!

However, I do not want to make an issue of the number of error events that we have to consider to fairly evaluate a candidate code.

We should focus more on the size of the dictionary versus the gain in code performance. The situation is reviewed as follows:

RATE ½ CODES

CODE SIZE	DICTIONARY ENTRIES	PROBABILITY OF ERROR		
		RANDOM CODE	BEST CODE	PRACTICAL CODE
4	4	.0304	.0212	.0212
8	16	.0251	.00786	.00786
16	256	(hard to compute)	.00295	.00368
32	65,536	(impossible to compute)	.00035	.0029
64	4 billion	(impossible to compute)	.000042	.00127

So far I have withheld a most interesting fact. When Shannon used randomly chosen code dictionaries to underbound the performance of the unknown, optimum codes, he found that these random codes on the average fulfilled the promise of the channel capacity theorem—the probability of error approached zero exponentially as the code length increased. This retrospectively confirms the intuition we gained from our example. Codes get better as they get longer. Furthermore, amazingly, it is not necessary to be very selective about the code dictionaries; merely picking them at random works perfectly well. Of course, as you can see from the table, if you are really able to pick an optimum (best) code, you do a bit better. But really, almost any old code will do. Or so it seems.

Right away there are two problems. First, we really do not know how to make optimum codes of any appreciable length. Second, it is apparent that even though picking random codes sounds simple in concept, we would have to deal with horrendous numbers to construct longer codes. A code length of 100 might be reasonable in practice, but for a code with an information rate of ½, just writing down the dictionary would require a thousand trillion entries, not even to speak of the decoding problem. Thus we cannot even keep a dictionary! Random codes are out. We must instead find a code that we can mechanize; a code for which the appropriate code words can be *calculated* upon demand by recipe, and similarly decoded. We call these codes constructive, as opposed to random. We cannot keep dictionaries, we must build machines.

Through the work of thousands of intrigued scientists and engineers over the last forty years many constructive coding procedures have become known. The tyranny of numbers is so overwhelming that a very strong structure, a sort of mathematical scaffolding, must be built into the coding algorithm. There must be a strong rationality underneath the way the code words are derived. In a moment I will describe a popular means of mechanizing the selection and generation of code words. For comparison with the random and optimum codes in the earlier table, I have included the performance of some practical codes that can be derived by mechanized procedure, and likewise decoded. Notice that these mechanizable codes have a quite reasonable error-rate performance, so it appears that there is little penalty for requiring practicality.

MECHANIZING A CODE

We need to find some kind of mathematical crank that can turn out code words on demand, and that will provide some foundation for analysis so that we can predetermine the code performance. Among the many possible binary sequences that could be code words, we need to pick some small fraction of words that are somehow "far apart." Suppose that we were dealing with real numbers instead of binary sequences. One way that might occur to you might be to pick, say, every sixth number—0,6,12,18 . . . . The distinguishing characteristic of the "code numbers" would be that each would be divisible by six. It turns out that we can do something very much analogous with the binary sequences by making every legal code word evenly divisible by something called a generator polynomial. Let me explain.

As an example let me return to the Hamming code that I mentioned before. Each code word is 7 bits in length, and since there are 4 information bits in this code, there are 16 code words that must be selected from the 128 possible 7-bit sequences. Let me represent a 7-bit sequence as a polynomial in a variable X with binary coefficients corresponding to the bits in the sequence. If a particular bit in the sequence is a 0, then that term in the polynomial disappears. (I am sure that worries some of you, but wait a little, it is just a simple and powerful convenience.) Here are some example sequences and their polynomial representations:

$$\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 1 & 1 & 1 & \\ X^6 & & & +X^3 & + X^2 & + X & + 1 & ; \end{array}$$

$$\begin{array}{cccccccc} 0 & 1 & 1 & 0 & 0 & 1 & 0 & \\ & X^5 & + X^4 & & & +X & . & \end{array}$$

That seems simple enough. For each 7-bit sequence we can write a corresponding polynomial in X, but why would we want to do that? Well, the secret behind the Hamming code is that each code word has a polynomial that is evenly divisible by the generator polynomial

$$g(X) = X^3 + X + 1.$$

For example, 0100111 is a legal code word in the Hamming code, and its polynomial representation is

$$\begin{aligned} X^5 + X^2 + X + 1 &= g(X)\ (X^2 + 1). \\ &= (X^3 + X + 1)(X^2 + 1) \end{aligned}$$

Try this multiplication yourself, but you have to use the rules of binary arithmetic on the coefficients. That is: $1 + 0 = 1$, $1 + 1 = 0$, $1 \times 1 = 1$, $1 \times 0 = 0$. Thus any code word is a multiple of that generator polynomial, and any sequence that is not a multiple of it is not a code word.

Now suppose someone says to you that they would like to transmit the information bits 0010. How do you find the right code word using the idea of the generator polynomial? In real numbers that would be like trying to find the multiple of 3 that was nearest (but not greater than) the number 22. We would first divide 22 by 3 and notice that we had a remainder of 1 left over, so by subtracting the 1 from 22 we would reach the desired answer of 21. We do exactly the same thing with the generator polynomial. First we shift the information bit sequence 3 positions to the left to leave room for the check bits:

0 0 1 0 _ _ _, or in terms of the polynomial, X^4.

Then we divide that X^4 by the generator polynomial. We get a remainder of $X^2 + X$, so by analogy if we subtract this from the X^4 the result will be evenly divisible by the generator polynomial.

In the binary field, addition is the same as subtraction, so the code word we find is:

$X^4 + X^2 + X$ or in terms of the bit sequence, 0 0 1 0 1 1 0.

So this is the simple recipe that produces code words from information sequences. The entire Hamming code generated by this process is as follows:

0000000	0001011	0010110	0011101
0100111	0101100	0110001	0111010
1000101	1001110	1010011	1011000
1100010	1101001	1110100	1111111

I would like you to take a moment to appreciate the beautiful structure in this code. I said it had to have very strong underpinnings, and you can see this by noticing the following properties of this code:

- Any sum of 2 or more code words equals another code word. For example, 0100111 + 1001110 = 1101001. Try it!
- Shift any code word by any amount in either direction, wrapping around the bits as they fall off the end, and you will get another code word. For example, 1100010 shifted right one bit is 0110001. Because of this property, the code is called cyclic.
- Every code word differs from every other code word in at least 3 positions. This is easiest to see by considering the code word 0000000. Notice that every other code word has at least 3 ones. This minimum distance of 3 guarantees that any single error will result in a sequence that is still closest to the original code word. Thus this Hamming code is called a single-error-correcting code. On the other hand, if it is used strictly for error detection, it takes at least 3 errors to turn a code word into another code word.

What marvelous properties! Where did they come from, you might ask? Partly they were the result of making the code from the multi-

ples of the generator polynomial, but to get that last property of good minimum distance (which equates to good error-correcting abilities), the right generator polynomial had to be picked to begin with. And where do you get a good generator polynomial? The truth is—you look it up in a table that has been put together by mathematicians who possess the powerful magic to know which polynomials are good and which are not. The mathematicians can tell the error-correction properties from the roots of the generator polynomial $g(X)$. The roots of $g(X)$ are the values of X for which it is zero, but unfortunately the roots are not nice numbers like 2 or 3, rather they lie in mathematical abstractions called Galois fields. Sorry about that, but there are really good tables available even if you do not know about Galois fields!

There are very simple electronic circuits or software algorithms that do the polynomial division needed to calculate the check bits. The polynomial that protected all of my precious bits in this book was $X^{16} + X^{12} + X^5 + 1$. This polynomial is a standard used in many common communications software packages. It produces 16 bits, or 2 bytes, of remainder after division into the polynomial corresponding to the desired block of information. These bits are usually called Cyclic Redundancy Checks (CRCs), for some forgotten reason. At the receiver it is simple enough to recalculate the CRCs with the same generator polynomial to see whether or not there is agreement. On the other hand, if you want to do error correction instead of merely detecting errors at the receiver, the algorithms that find the maximum likelihood data sequence are really quite intricate and cumbersome. They do not strike me as so beautiful and shiny as the code generation algorithms, but fortunately there is always someone crawling under the engine who is really into these things.

THE CODING PARADOX

As constructive coding procedures like the one we just described were uncovered, their inventors gathered mathematical arguments proving the value of their codes. An obvious question concerned the power of these codes to decrease the error rate towards zero while holding the information rate constant. Surprise! The con-

structive codes were asymptotically bad, that is, as the block length became long either the information rate went to zero or the error rate approached .5 (complete garbage). This provoked consternation; codes picked entirely at random were good, but codes picked according to the most ingenious mathematical procedures were failures. The conundrum was aptly stated as, "All codes are good, except those that we know about." In 1972 a class of constructive codes was discovered that does achieve arbitrarily small probability of error at nonzero information rates (though not as high as capacity). Although this class is of philosophical interest, it has not been practically important. With this exception, it is still true that though any old code should do, essentially all the ones we know about do not have the asymptotic behavior that Shannon's results promise. Nevertheless, we do have very powerful coding algorithms for a wide range of midrange applications.

How can it be that randomly chosen codes have desirable properties, while mathematically chosen codes do not? It seems that what we are really after is true randomness, which is difficult to emulate with mathematical structure. What makes codes good is that the code words are far apart from each other. In a sense we want to sprinkle the code words throughout the available space uniformly in order to keep them far apart. Imagine being keeper of the universe and having to distribute the stars in the heavens—except that the space is not the three-dimensional, physical world but many-dimensional and binary. (We humans do have a way of visualizing things in our own spatial experience, and in some coding applications code sets are called "constellations.") In any event, one way to get a good distribution of our stars is just to throw them out there randomly. But unfortunately there are just so many that we cannot keep track of them all! On the other hand, if we derive ingenious, but relatively simple ways of accounting for and positioning our stars, they do not spread out uniformly but tend to cluster in some sense. Can we emulate randomness with an underlying mathematical structure? We do not know that it is impossible, but it does appear to be difficult.

Now I should like to explain why the error rate can be made to approach zero as the code length is made longer. The reason is quite fundamental and is shared with many physical phenomena; it is the statistical reliability of large collections of random numbers. Shannon's promised behavior relies on the same phenomenon that guarantees insurance companies a certain rate of return despite the

probabilistic nature of their business. Many of us know these laws of large numbers from courses in statistics and probability. If I flip a coin once, I may get a head, or I may get a tail. There is not much we can say about the result *a priori.* If I flip the coin ten times, I would be surprised if I got less than 20 percent (i.e., two) heads, but not terribly so. If I flip the coin a thousand times, I would expect that the ratio of heads would be almost exactly .5. I do not expect exactly 500 heads, which is improbable, but I expect the *fraction* of heads to be very close to .5. For example, there would be high likelihood that between 470 and 530 of the 1,000 flips would be heads. The more I flip the coin the greater the probability that the ratio is close to .5. This is assured in spite of the fact that each individual toss is itself unpredictable. The standard deviation of the percent of heads shrinks as the inverse of the square root of the number of tosses.

This same predictability occurs in the transmitted data bits. With an error rate of .02, we should expect that as the code length becomes longer, very close to 2 percent of the bits will be in error—not 1 percent or 3 percent, but more and more exactly 2 percent. All we need is a code that corrects error patterns that deviate 2 percent in distance from the original code word. If we can make such a code, it will correct essentially all of the errors. A somewhat more informative way of looking at this phenomenon was pointed out by Shannon. Consider all the error patterns of length equal to the word length N. There are obviously 2^N of them, and for long word lengths that is a huge number of situations for the code to have to worry about. However, Shannon showed that only the smallest, minute fraction of these possible sequences becomes of any concern whatsoever as the length N becomes large. Specifically he found that only 2^{HN} sequences are significant: These sequences contain virtually all of the probability, while all other sequences have probability tending to zero. Thus the fraction of significant sequences is $2^{HN}/2^N = 2^{-CN}$. For example, in our channel with probability of error .02 and channel capacity $C = .8585$, if the code length is 100 then the fraction of error sequences that we have to worry about is only about 2^{-86}. This is the fraction of sequences with 2 percent of the bits in error. I do not have to tell you that that is an incredibly small number.

I would like to elaborate somewhat on this fundamental behavior. Here again we see that the entropy H is a central quantity. It represents the fraction of the *bits* that is significant in describing

error locations. In our example, where $H = .1415$, the uncertainty in the error locations is low enough that the locations of the errors can be described by about 14 percent of the total bits, so that only $2^{.14N}$ error patterns occur with significant probability. All other error patterns have a very small total probability. In other words, certain error patterns are "typical," while all others are "atypical." As the length of the pattern considered becomes very long, the "typical" error sequences predominate.

Imagine that we list all of the 2^N possible error patterns in order of probability. The first such pattern is 0000 . . . 00, indicating no errors, while the next N patterns contain a single error in one location, such as 0000 . . . 01, etc. The next $N(N-1)/2$ patterns are the ones with two errors, etc. If we draw a line horizontally across our list after the $2^{HN}th$ entry, then all of the patterns above our line are typical. All the patterns below are atypical, and as the length N becomes large, these latter simply do not occur. (More properly, they occur with vanishingly small probability.) It is wonderful how so much certainty can be associated with the overall properties of random events!

If we return to our analogy of the coding problem as one of distributing stars in a universe, the significance of the existence of "typical" error sequences has a geometric interpretation. The code word positions are where the stars are located, but during transmission, error patterns that alter their locations are added to them. However, as we have seen, the disturbance does not take the stars just anywhere, but it only moves them to some point (in our example) that is approximately 2 percent away in bits. Thus a received star will lie somewhere roughly on a sphere centered about its original location. As the sequence gets longer and longer, we have the same geometric interpretation, except that the dimensionality gets higher and higher, and because of the law of large numbers the "skins" of the spheres where the stars are most likely to be found get thinner and thinner. With a high probability, the received sequences are found nearly on the surface of those 2 percent-deviation spheres. This is a phenomenon known as "sphere hardening." It is also a familiar fact in mathematics that all of the volume of a sphere goes towards its surface as the dimensionality gets higher and higher.

Because of sphere hardening, we know more and more exactly the region in which the received sequences will lie. If the code is ready for them, the errors will virtually all be corrected. In con-

structing a code, as in distributing the stars, all we have to do is to be sure that none of those spheres centered on our code words intersects with any other such sphere centered on a different code word. As we have seen, if the code rate is less than capacity, and we just throw the stars out there, it works—at least in the limit of long codes. When we examine the possible performance of finite-length codes, one of the methods for finding bounds on performance is termed "sphere packing." Just how many spheres of a given diameter can we fit into the space without overlap? How many Ping-Pong balls will fit inside a beach ball? Answers to these questions, which are the kinds of problems that fascinate mathematically inclined researchers, can tell us just how well a code of a given length and rate can do.

Arguments such as sphere packing help tell us just how long a code has to be to achieve a given error rate performance. After all, we are caught on the horns of a dilemma; the error rate can only be driven down by the reliability of the statistics of large numbers, thus we need long code lengths to encompass a significant number of error events. On the other hand, as the code length becomes longer, the number of code words, and consequently the complexity of the encoding and decoding process, increase exponentially. From the viewpoint of code derivation we are concerned with how many spheres can be packed into a given space, and at the same time as the code becomes longer, we await anxiously the effect of the law of large numbers causing the surfaces of the spheres, representing the probability densities of the received words, to "harden" into the Ping-Pong balls we hold in our imagination.

If we hold the effective information rate of the code constant while we increase the length N, the error rate attainable with the best possible code will decrease exponentially, that is,

$$Prob(error) < e^{-rN}$$

What interests us the most is the value of the exponent r, since that will determine how long a code must be for a given level of performance. Through the years a number of ingenious arguments have been used to find upper and lower bounds on r. It has been shown that the closer the information rate is to the channel capacity C, the smaller r becomes, and consequently codes have to be very long to be effective. The implication of this fact is that codes get better quickly when we operate at a small fraction of

capacity (this is why we were not greedy in our example), but that they must be long when we try to get near capacity. What makes the situation worse is that long constructive codes may not be very efficient. Thus the story behind Shannon's remarkable theorem is not as sanguine as the simple statement of the theorem might lead us to believe.

A RETROSPECTIVE ON INFORMATION THEORY

Some of the key ideas in information theory that we have discussed are the following:

- Information can be measured by the logarithm of the probability.
- The communications system can be divided into source coding to eliminate needless redundancy, and channel coding to incorporate desirable redundancy.
- There is a fundamental channel capacity that measures the maximum information rate a communications channel can sustain. The concept of channel capacity bridges the two concepts of source coding and channel coding. If the capacity of the channel exceeds the information rate of the source, then the source can be transmitted exactly over the channel.
- If the information rate is less than channel capacity, then error-correcting coding can be used to reduce the error rate to any arbitrarily small number. The error rate decreases exponentially as the code length becomes longer. At the same time the complexity of the system becomes exponentially larger.

When we examine such forms of information as text, speech, images, and computer files in future chapters, we shall have occasion to apply an information-theoretic viewpoint from time to time. It will serve sometimes as a philosophy, and at other times as an

inspiration of what might be possible. Information theory can serve to anchor our feet, and to tell us when it would be foolish to dream of further gains. The core of information theory that we have presented here is a very beautiful work of human thought. In the ever-changing tapestry of the computer age it is one of the few timeless subjects. It is a classic tale, retold many times, yet it is with enthusiasm that I include my own version of the essentials here in the context of information technology.

When information theory was first publicized, it achieved immediate popularity and was applied to many fields of human endeavor. The literature was filled with attempts to use the principles of information theory in psychology, art, and music. These applications were often disappointing and are seldom seen today. Instead, the pendulum has swung the other way, and workers in the field are often skeptically asked what, if anything, information theory has accomplished. There are perhaps three answers to such a question. First, information theory stands alone as a work of art that needs no rationalization for its preeminent place in human knowledge. Second, information theory has served as a guiding philosophy in the design of communications systems. Lastly, information theory has led both directly and indirectly to the invention of many classes of useful error correcting and detecting codes. Only now that computation has become relatively inexpensive are these codes beginning to see widespread use in communications and in data storage and retrieval. If information theory is regarded as having had minimal impact on the world, it is because of unreasonable expectations and a lack of understanding of the depth and wisdom it contains.

REFLECTIONS—CODING IS DEAD

There is a small group of us in the communications field who will always remember a workshop held in Florida about twenty years ago at which we engineers took a look at the future and saw nothing but gloom for technology. One of my friends gave a talk that has lived in infamy as the "coding is dead" talk. His thesis was that he and the other coding theorists formed a small, inbred group that had been isolated from reality too long. He

illustrated this talk with a single slide showing a pen of rats that psychologists had kept in a confined space for a long time. I cannot tell you here what these rats were doing, but suffice it to say that this slide has since been borrowed many times to depict the depths of depravity into which a disconnected group can fall. The rats made *Lord of the Flies* look like a school picnic.

All this depraved behavior had its parallel in the activities of the coding theorists, he argued. Too many equations had been generated, with too few consequences, just for the thrill of it all. One paper after another had justified itself on the basis of being an extension of the results of a "famous" predecessor paper, which unfortunately nobody outside the closed group cared about. (I am reminded of a proposal I was asked to review that began, "Lately there has been a great deal of interest in . . ." citing references 1 through 15. Checking in the back I discovered that those references had all been written by the author himself.) Coding theorist professors had begat more coding theory Ph.D.s in their own image, all preprogrammed with reverence for the mythical Galois fields, as a sort of mathematical Stonehenge. But no one else cared; it was time to see this perversion for what it was. Give up this fantasy and take up a useful occupation, exhorted my friend. Coding is dead.

Carried away with the mood of this somber day, I made my own prediction. "Data is dead," said I. There was an immediate chorus of dissent. It was not right to say this, they shouted. One should say instead "data *are* dead," in order to properly reflect the plural form. I stood corrected, but nonetheless went on to bemoan the coming of the dreaded digital networks, which would obviate all need for modems to connect to analog facilities. "Modems are dead," I said, confident of my grammar. Heads nodded in unison. Scratch modems, no point in working on them anymore. Marvelous how we technologists could chart the future when we put our minds to it.

I am sure there are morals enough in this little vignette for many an essay. Fortunately, no one gave up coding or modems. Today ads for modems appear on national television during football games. At home I have a Reed-Solomon coder casually sitting on an end table in my living room. It is inside a compact disk player, of course. Otherwise it would look rather silly. The computer in my basement uses error-checking hardware in its memory, and cyclic codes for disk access. And, yes, when I use my modem to send files to the office, codes are applied once again. My children even have their own modems. Great predictions, weren't they?

Why are we technologists so bad at predicting the future of technology? Now I know that futurology is easy to discredit. Famous wrong predictions are legion; it would not even be sport to quote them here. But it seems to me that we technologists are less adept at looking at the future of

technology than are laypeople. The science fiction writers claim to have foretold almost everything we do, and in some cases, like Arthur C. Clarke and the communications satellite, there seems justification. Of course, from where we sit their job looks easy—throw out an infinite number of arbitrary predictions, and many will come true. Moreover, they are not burdened with knowing why it is impossible to do the things they suggest, and this is our own handicap. We are too close to the technology itself. From our myopic viewpoint we see only the problems, and not the possibilities that transcend what others see as only "engineering" details. Then, to be honest, some of our detractors say we lack imagination. Better than lacking breeding, however.

Of course most of the public thinks of us as being in charge of the future. When we make predictions, they listen. Silly them. If the coders had listened to themselves, they would have given up coding, but secretly they knew better. They would never have listened to their own predictions. But others have no such qualms. If the newspapers had covered our little meeting, I can see the headlines that would have resulted. "Data is dead, say experts." (Everyone quoted in the newspapers is an expert.) In my company the people in charge of bureaucracy send me an annual demand for a list of "expected breakthroughs." All the researchers think as I do that this is absurdity beyond belief, but how can we reject the demand without conveying the idea that we do not know what we are doing? I tried to make it go away, but that was more trouble than filling it in with such preposterous predictions as "room-temperature superconductivity." I suppose there are advantages to being thought of as omniscient.

That our predictions are usually worthless is but one interpretation to draw from this little incident. Let me give you a choice of morals to try on.

1. Coding theorists prove once again that engineers cannot predict future.
2. Coders persevere in spite of skepticism and, believing in their own work, triumph in the end.
3. Support fundamental studies in math. Do not be shortsighted, they will pay off in the long run.
4. Lazy theorists, comfortable in their work and unwilling to find new fields of endeavor, stick with the only thing they know and somehow luck out.

I do not mean to pick on coding theory. The same thing could be said about any field that has academic roots. Some we win, some not, but again and again we are faced with deciding what work we stop and what

work we support, both for ourselves and for our businesses. While most people may feel that these decisions are especially difficult, I have no problem myself. I am fortunate in having this little list of things that are currently dead. Lack of space prevents reproducing the list here.