Communication on Noisy Channels: A Coding Theorem for Computation

Leonard J. Schulman*
Department of Mathematics
Massachusetts Institute of Technology
Cambridge MA 02139

Abstract

Communication is critical to distributed computing, parallel computing, or any situation in which automata interact — hence its significance as a resource in computation. In view of the likelihood of errors occurring in a lengthy interaction, it is desirable to incorporate this possibility in our model of communication.

In the noisy channel model for communication complexity, two automata are to reliably solve a problem to which each has only part of the input, by communicating across a faulty channel. This contrasts with the standard model for communication complexity, introduced by A. Yao, in which the channel is noiseless (does not introduce errors).

Noisy communication channels were first considered by Shannon in his seminal work in information theory. Shannon considered the "one-way" problem of transmitting a large block of data over a noisy channel. He described the intrinsic capacity of a channel and established his coding theorem, which states that the data can be encoded for transmission in such a way that the transmission time is proportional to the quantity of data and inversely proportional to the channel capacity, while a vanishing probability of error is incurred in the decoding.

We relate the noisy channel and the standard (noiseless channel) complexities of a communication problem by establishing a "two-way" or interactive analogue of Shannon's coding theorem: every noiseless channel protocol can be simulated by a private-coin noisy channel protocol whose time bound is proportional to the original (noiseless) time bound and inversely proportional to the capacity of the channel, while the protocol errs with vanishing probability.

Our method involves simulating the original protocol while implementing a hierarchical system of progress checks which ensure that errors of any magnitude in the simulation are, with high probability, rapidly eliminated.

*Support for this research was provided by an ONR graduate fellowship, an MIT Applied Mathematics graduate fellowship, and grants NSF 8912586 CCR and AFOSR 89-0271.

1 Introduction

The study of message transmission over noisy channels was pioneered in Shannon's work of 1948 [21]. Some of his fundamental contributions may be outlined as follows: (a) Message sources are usefully modeled as continual stochastic processes; the "rate" with which such a source produces information is measured by its entropy. (b) Communication channels have an intrinsic "capacity" which measures how much information they can carry in each transmission. (c) Messages may be transmitted over the channel at rates arbitrarily close to capacity (but no more), with arbitrarily low probability of error. The fundamental technique used to approach this objective, is the joint encoding for transmission of increasingly long input blocks.

Information theory, as taken to mean the study of information sources, communication channels, and the problem of message transmission, has been studied extensively since Shannon's seminal paper. In recent years however, communication has come to be investigated from a very different perspective, within computer science. A. Yao introduced the following model [26] of communication: The argument (input) of a function is split between two processors A and B. The processors are provided with a noiseless communication channel which can transmit one bit at a time in either direction. Their goal is to compute the value of the function, by communicating with each other using some agreed-upon protocol. The protocol can terminate when both parties know the value of the function. The primary object of interest is the *communication complexity* of the function: the number of bits that must be exchanged in the protocol, in order to solve the problem. (Measured on a "worst" input; often an average is also considered).

Shannon's work focused on the *transmission* of information: and he was able to say first, what is infor-

0-8186-2900-2/92 $3.00 © 1992 IEEE

mation, and second, what is a channel. In our time the telephone or telegraph have been joined by communicating, interacting computing devices: for example in integrated circuits, distributed networks of computers, and robotic control of remote devices. Communication complexity focuses on the two-way *exchange* of information, and is an attempt to understand computation.

The distinction is not primarily in the communication apparatus — much work has focused on the potential of feedback channels for assisting in message transmission — but in the problem. A two-way or interactive problem may depend on the inputs at both processors. One-way or transmission problems are the special case when the problem depends on only one of the inputs.

The critical role of communication in distributed and parallel computing, has motivated considerable interest in the study of communication as a computational resource. (Beginning with [26, 27, 14, 23, 12]; and see [13] for a survey). In view of the likelihood of transmission errors occurring in the course of lengthy communications, it is desirable to incorporate this possibility in our study of communication complexity.

In this paper we present one step in this direction: an analogue of Shannon's coding theorem for interactive communication problems.

The correspondence between Shannon's theorem and ours is as follows. On a noiseless channel, capable of carrying one bit per transmission, processor A can, in n transmissions, send processor B a block of n bits. Now fix a channel ("binary, symmetric") of capacity C ($0 \leq C \leq 1$). Shannon's theorem shows how the noiseless-channel procedure can be simulated on the noisy channel: i.e. how, for any small γ, A can encode the n bits into $n\frac{1}{C}(1+\gamma)$ channel transmissions, from which B can correctly decode A's data, except for an exponentially small failure probability $e^{-\theta(n)}$.

In our interactive analogue we consider *any* communication problem for which the processors have a noiseless-channel protocol π, which terminates within n transmissions on every input to the processors. We show how to simulate π on the noisy channel in $\theta(n/C)$ transmissions, except for an exponentially small failure probability $e^{-n^{1-o(1)}}$.

As in the one-way case, the computational resources of time and space available to each processor will be unrestricted. Unlike the one-way case, each processor will also have access to a private source of random coins. Our method involves simulating the original protocol while implementing a hierarchical system of progress checks, which ensure that errors of any magnitude in the simulation are, with high probability, rapidly eliminated.

2 A Coding Theorem

We begin by specifying a model of a noisy communication channel. The binary symmetric channel is a synchronous communication link which, in every unit of time, accepts at one end either a 0 or 1; and in response produces either a 0 or a 1 at the other end. With some fixed probability ϵ the output differs from the input, while with probability $1-\epsilon$ they are the same. This random event for a particular transmission, is statistically independent of the bits sent and received in all other channel transmissions. (Thus the channel is "memoryless".)

The average mutual information between two ensembles $\{P(x)\}_{x\in X}$ and $\{P(y)\}_{y\in Y}$, which have a joint distribution $\{P(xy)\}_{x\in X, y\in Y}$, is defined as $\sum_{xy} P(xy)\log\frac{P(xy)}{P(x)P(y)}$. This is a measure of how much information is provided about one ensemble by the specification of a point (chosen at random from the joint distribution) of the other ensemble.

The capacity of a channel was defined by Shannon as the maximum, ranging over probability distributions on the inputs, of the average mutual information between the input and output distributions. In the case of the binary symmetric channel the capacity (in base 2) is $C = 1 - \epsilon \lg \frac{1}{\epsilon} - (1-\epsilon)\lg\frac{1}{(1-\epsilon)}$. Observe that $0 \leq C \leq 1$; and that a noiseless channel has capacity $C = 1$. Gallager [9] provides a thorough exposition of these concepts.

In the standard (noiseless) communication complexity model, the argument (input) $z = (z_A, z_B)$ of a function $f(z)$ is split between two processors A and B, with A receiving z_A and B receiving z_B; the processors compute $f(z)$ (solve the communication problem f) by exchanging bits over a noiseless link between them. Each processor has unbounded time and space resources. Randomness may be allowed as a resource, and is typically considered at one of three levels: none (deterministic protocols); private coins (each processor has its own unbounded source of random coins, but cannot see the other processor's coins); and public coins (the processors can see a common unbounded source of random coins.) The procedure which determines the processors' actions based on their local input and transmissions received, is called a communication protocol.

In the present paper the noiseless link is replaced (in each direction) by a binary symmetric channel of capacity C.

Here is a precise statement of Shannon's coding the-

orem. (Not in its most general form. See [21]; [9], theorem 5.6.4).

Theorem 1 (Shannon) *Let a binary symmetric channel of capacity C be given. For every n and every $\gamma > 0$ there exists a code $\chi : \{0,1\}^n \to \{0,1\}^{n\frac{1}{C}(1+\gamma)}$ and a decoding map $\chi' : \{0,1\}^{n\frac{1}{C}(1+\gamma)} \to \{0,1\}^n$ such that every codeword transmitted across the channel is decoded correctly with probability $1 - e^{-\Omega(n)}$.*

If all one wants is to transmit one bit (or a fixed number of bits) and a very small probability of error is desired, then there is only one thing to be done: the redundancy of the transmission must be increased, in order to drive down the error probability. (By redundancy we mean the number of transmissions per bit of the message). Shannon's theorem says that this is not necessary if one may assume that the message to be transmitted is long. In that case codewords (the images of n-bit input blocks via χ, as above) can be selected in such a manner that when any one of them is transmitted over the channel, the probability that it so changed by the channel noise as to be mistaken by the receiver for another codeword, is exponentially small in n. Since the exponential error is gained from the block size, n, which may be chosen as large as desired, it is not necessary to invest extra redundancy for this purpose. It is enough just to bring the redundancy past the threshold value $1/C$.

Following is an analogue of Shannon's theorem for the more general case of two-way, or interactive, communication.

Theorem 2 *In each direction between the processors let a binary symmetric channel of capacity C be given. There is a constant σ such that for every n, every $\gamma > 0$ and every communication problem f with a noiseless channel protocol π of length n, there exists a private-coin noisy channel protocol of length $n\frac{\sigma}{C}(1+\gamma)$ which for every input pair, solves f correctly with probability $1 - e^{-n^{1-o(1)}}$.*

There is a point to be noticed here. In a general protocol π, there are *no large input blocks.* In the nature of interaction, processors generally do not know what they want to transmit more than one bit ahead. Thus the methods used for the transmission problem do not suffice for the interactive problems.

A naive simulation of the noiseless channel protocol π might involve repeating every character transmitted in π, k times. The receiving processor would decode the transmission by taking a majority vote among the k characters received. The probability of an error occurring in such a transmission is $e^{-\theta(kC)}$, where C is the capacity of the channel.

However, if any of these transmissions fails, the remainder of the protocol is wasted, as each processor will be laboring under a misunderstanding.

Thus in order to assure even a constant or polynomial probability of error, the redundancy needed is nonconstant, $k = \theta(\frac{1}{C}\log n)$. Further in order to assure an exponentially small probability of error, as we desire, this naive simulation needs polynomial redundancy $k = \frac{1}{C}n^{\theta(1)}$.

We will prove the coding theorem by using a more sophisticated simulation of π. In section 4 we describe a public-coin protocol to simulate π, and in section 5 we analyze that protocol, showing how it avoids sacrificing efficiency for reliability. In section 6 we show how to transform the public-coin protocol into one that uses only private coins.

First however we describe the notion of progress checks. They will play an important role in our solution, although as we shall see, they are not alone sufficient.

3 Progress Checks

Here is a possible general strategy: the processors alternate rounds of simulation of π, with "progress check" rounds in which they try to determine whether the simulation is proceeding correctly; and if not, they backtrack in the simulation until the checks show that they have undone the error, at which point they resume the simulation.

At time t, processor A holds a binary vector, rec_A, of length t, representing its record of the protocol to date: these are the bits transmitted from A to B (before repetition), as well as the bits A has decoded from B's transmissions. Similarly B holds such a binary vector, rec_B. Observe that the simulation to date is correct if, and only if, $\text{rec}_A = \text{rec}_B$. Thus a progress check might be implemented by having the processors compare the vectors rec_A and rec_B.

These vectors can be quite long: t ranges up to n. Even for a noiseless channel, A. Yao has shown [26] that deterministically comparing two vectors of length t requires $\Omega(t)$ transmissions. However, using randomness, a comparison can be implemented in far fewer transmissions: A *constant* number of transmissions suffice in order to compare a pair of arbitrarily long vectors, with only a small constant failure probability. Specifically, let R be a $t \times k$ matrix with entries independently and uniformly selected in $GF(2)$.

Lemma 1 *If $rec_A \neq rec_B$ then with probability $1 - 2^{-k}$, $rec_A R \neq rec_B R$. If $rec_A = rec_B$ then $rec_A R = rec_B R$. (Operations in $GF(2)$.)*

Proof: Standard. □

In the public-coin model, both processors can see the same random matrix R. They can implement a progress check by first using R to calculate a k-bit parity-check vector of their own record and then using a deterministic Shannon code to encode the parity-check vector into $\theta(k/C)$ channel transmissions. Except for a failure event of probability $e^{-\Omega(k)}$, each processor will correctly decode the other's parity-check vector.

There are two ways in which this procedure can fail. First, the random matrix R might map two different *rec* vectors to the same parity-check vector — resulting, in case of a successful channel transmission, in the processors mistakenly believing the simulation is correct. This is not a problem: the difference between their *rec* vectors will persist, and most likely they will soon discover it and backtrack in the simulation.

The second way in which this check might fail, is through channel errors during the checking procedure. If both processors suffer from the error, then this is no worse a problem than the first kind: in case of a "false equality" the processors lose a round before backtracking, while in the case of "false inequality", a round of the simulation is needlessly erased.

A more serious problem is that channel errors during the checking procedure will occasionally cause one processor to believe the simulation is correct, and therefore continue in the simulation; while the other believes there is an error, and backtracks a step. In this situation the processors differ in their notions of the simulation step t (i.e. in the positions of their "pointers" into the noiseless protocol), and necessarily therefore their *rec* vectors differ[1]. Henceforth in most rounds, the processors will be engaged in backtracking in order to undo the error which they detect. Unfortunately this does not help them close the gap between their pointers: in fact the only way the gap can be closed, is by further asymmetric errors which cancel the effect of the first. Since the asymmetric errors do occur with some constant frequency, the gap between the pointers will undergo an unbiased random walk. The gap starts at just one unit, and so with very high probability it will in fact vanish within a constant number of rounds; moreover the probability of it eventually vanishing is 1. However: the *expected* time until the gap vanishes is infinite. This is a classic phenomenon in gambler's ruin ([5] §XI.3, XIV). In terms of the protocol, it means that the simulation will completely unravel.

Roughly speaking, the above simulation was too homogeneous. In the next section we will describe a solution which assists in bringing the processors' pointers together by funnelling them through a hierarchy of prearranged "meeting points".

4 Public-Coin Simulation Protocol

In this section we describe a public-coin protocol for theorem 2. We defer the private-coin case to section 6.

We distinguish two notions of time. *Step t* will refer exclusively to the t^{th} bit in the noiseless protocol π. *Round t* will refer exclusively to the t^{th} occasion on which the noisy simulation simulates a step of π. Due to the synchronicity of the model, the processors always know and agree on what the current round is; on account of channel errors they might, however, differ as to what step they are currently simulating. We will describe each processor as having a pointer, whose value is equal to t during a certain round if the processor is then trying to simulate step t of π.

We will assume without loss of generality that in the noiseless protocol π the processors alternate single-bit transmissions; this is the "hardest" case.

If π is a protocol of complexity n, the processors will run the simulation for $N = 2^{\lceil \lg n \rceil + 1}$ simulation rounds. (Note that $2n \leq N < 4n$.) Each round will consist of one of the processors simulating a single step of π, by repeating that bit k/C times (for some constant k)[2]. In between rounds the protocol will also execute a hierarchical system of progress checks, each associated with an internal node of a complete binary tree, according to the following schedule. Checks will be labeled by their level in the tree (the root being at the highest level). The actual rounds of simulation will be associated with the leaves of the tree (level 0). The protocol is scheduled in "postorder" on this tree: that is, a level l check will be executed after first its left subtree, and then its right subtree, have been fully executed (see figure 1).

We now describe the progress checks of the protocol. Recall that errors may result in a separation between the processors' pointers, as well as in differing bits in the records. The progress checks are designed

[1] The linear test we have described will not distinguish between a vector and some extension of it with only 0's at the end. This is easily remedied, for instance by augmenting the *rec* vectors with a representation of the pointer value t.

[2] Note that due to redundancy in the implementation of the simulation rounds, and to progress checks intervening between the rounds, the elapsed simulation time will be greater than the round number. However they will be the same to within a constant factor.

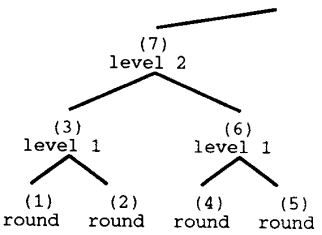

Figure 1: Simulation schedule

to deal with any combination of these events. If A's pointer is at step t_A, and B's pointer is at step t_B, then we define the current **position** of the protocol to be $\max\{t_A, t_B\}$. The current **gap** of the protocol is the difference between the position, and the end of the prefix of agreement of the two records (i.e. the greatest t such that the first t bits of rec_A agree with those of rec_B). The gap is the distance that one of the processors must backtrack, and thus a measure of the severity of the current error in the simulation. The simulation is correct after some round precisely if the gap is 0.

Performing a level-l check

A level-l check is performed as follows. Processor A writes its position t_A in the form $c_1^A 2^l + c_2^A$ (for $0 < c_2^A \leq 2^l$), and forms the three vectors of length N:

- $a_1 = \text{rec}_A$ restricted to positions $\{1, ..., (c_1^A - 1)2^l\}$. (Set a_1 to 0 in $\{(c_1^A - 1)2^l + 1, ..., N\}$).
- $a_2 = \text{rec}_A$ restricted to positions $\{1, ..., c_1^A 2^l\}$. (Set a_2 to 0 in $\{(c_1^A - 1)2^l + 1, ..., N\}$).
- $a_3 = \text{rec}_A$. (Set a_3 to 0 in $\{(c_1^A - 1)2^l + 1, ..., N\}$).

Similarly B forms b_1, b_2, b_3 from rec_B.

Now the processors observe a common random $N \times \theta(2^l/l^2)$ matrix R. A encodes each of the parity check vectors a_1R, a_2R, a_3R into $\theta(2^l/l^2C)$ channel transmissions using a deterministic Shannon code, and sends the three codewords to B. Similarly B sends encodings of b_1R, b_2R, b_3R to A.

The construction should be understood in this way. The positions $(c_1^A - 1)2^l$ and $c_1^A 2^l$ are places at which A proposes the two processors meet, in case any recent errors have occurred in the simulation. (The comparison of a_3 with b_3 is meant to detect such errors.) B also proposes two meeting places, $(c_1^B - 1)2^l$ and $c_1^B 2^l$. The key to the processors' success is that, so long as the gap is no more than 2^l, they will always propose at least one common meeting point, and share a common history of the protocol through that point. So they can agree to shift their pointers to that point and resume the simulation from there.

This idea is implemented as follows. Each processor represents the information available to it as a bipartite graph. On one side are three vertices representing A's three vectors a_1, a_2 and a_3. On the other side are three vertices representing b_1, b_2 and b_3. In A's graph, an edge connects a_i to b_j if a_iR equals the decoding of the transmission of b_jR. Similarly in B's graph, an edge connects a_i to b_j if b_jR equals the decoding of the transmission of a_iR. Thus edges of each graph represent presumed equivalences between the a_i and b_j vectors.

Observe that in any state of the simulation, there is some underlying correct graph representing which of the pairs a_i, b_j are indeed equal. We will classify checks as "good" or "bad":

Definition *A check is* **good** *if it results in both processors drawing the correct graph. Otherwise the check is* **bad**.

A check may be bad due either to channel errors or to a poor choice of R (creating spurious equivalences between the parity check vectors).

Each processor acts, depending upon its graph, as follows (description for A; everything for B is symmetric):

Case 1 A's graph is a "parallel" matching with three edges (see figure 2).

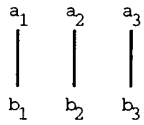

Figure 2: Graph for case 1

This suggests to A that the simulation is correct. So A continues simulating π from its current pointer position.

Case 2 A's graph is one of the "partial parallel" matchings in figure 3.

This suggests to A that, although an error exists in the simulation, the prefix of agreement nevertheless persists through a_1, a_2 or a_3 (depending on the case). So A moves its pointer back to the latest of a_1, a_2, a_3 that is incident to an edge, and

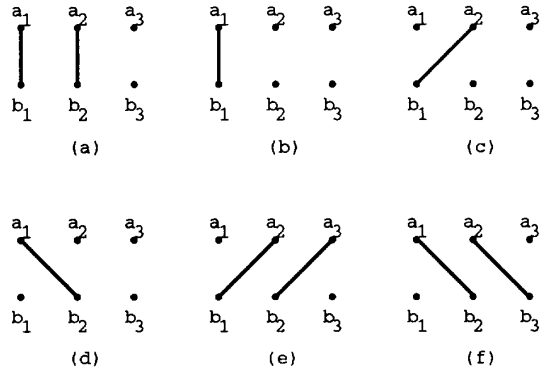

Figure 3: Graphs for case 2

continues the simulation from there. (I.e to a_2 in (a); a_1 in (b); a_2 in (c); a_1 in (d); a_3 in (e); and a_2 in (f)[3].)

Case 3 A's graph is empty. This suggests to A that there is an error in the simulation and that no a_i and b_j are equal. (Thus that the gap is greater than 2^l). So A does nothing. (Which means that it keeps its pointer fixed until a future check indicates otherwise. If there is a simulation round immediately after the check, A can send some default message, but it does not advance its pointer.)

Case 4 If A's graph is anything not described above, then the check must have been bad. So A does nothing (which means the same as in case 3).

Properties of a level-l check

Let us specify what a level-l check accomplishes.

If the check is bad, as defined above — that is if one of the processors draw a graph different from the correct one — then the check will not cause a loss in position, or an increase in gap, of more than 2^{l+1} steps.

On the other hand if the check is "good" then:

1. If the gap is 0: Both processors will know this, and will continue in the simulation.
2. If the gap is nonzero but bounded by 2^l: the check will enable the processors to eliminate the gap while incurring a loss of at most 2^{l+1} in the position.
3. If the gap is greater than 2^l: Both processors will know this, and neither will move its pointer.

[3] Cases (e) and (f) are unusual. Case (e), asserting $a_3 = b_2$, can correctly occur only if $c_2^A = 2^l$. Similarly (f) asserting $b_3 = a_2$ can correctly occur only if $c_2^B = 2^l$.

5 Analysis

Observe that if the simulation protocol is run for $N = \theta(n)$ rounds, the total transmission time will be $\theta(n/C)$, as we desire. For, each level l has $\theta(N2^{-l})$ nodes, each using $\theta(2^l/l^2C)$ transmissions. The total number of transmissions is thus $\theta(\sum_l N/l^2C) = \theta(N/C) = \theta(n/C)$.

We now restate the coding theorem for the case of public coins, and prove it. Since we will not determine the true value of σ, the θ notation will suffice in place of $\sigma(1+\gamma)\ \forall\gamma > 0$.

Theorem 3 *Let a binary symmetric channel of capacity C be given. For every n and every communication problem f with a noiseless channel protocol π of length n, there exists a public-coin noisy channel protocol of length $\theta(n/C)$ which for every input pair, solves f correctly with probability $1-e^{-n^{1-4\log\log n/\log n}}$.*

Proof: We analyze the protocol in two stages. First, we will assign a certain "bad weight" to every bad node of the simulation tree, and show that with high probability, the total bad weight of the simulation is small (bounded by a constant fraction of n). Second, we will show that a simulation with small bad weight completes π correctly.

We have already defined bad checks (internal nodes of the simulation tree) as those in which some processor draws an incorrect graph. Now we also define what it means for a round to be bad. Recall that each round of the simulation is concerned with simulating the transmission of a single bit of the noiseless protocol.

Definition *A round of the simulation (leaf of the simulation tree) is* **good** *if the character decoded equals that transmitted. Otherwise the round is* **bad**.

We will assign bad weight 2^l to a bad node at level l (e.g. a bad round has weight 1). The total bad weight of the simulation, denoted B, is the sum of the bad weights of its nodes. In view of the fact that $N \geq 2n$, the theorem is implied by the following two claims, whose proofs will be presented in the remainder of this section.

I. *With probability $1 - e^{-\theta(N/\log^4 N)}$, $B \leq N/34$.*
(With high probability the bad weight is small.)

II. *The simulation completes at least $N - 17B$ steps of π correctly.*
(A simulation with small bad weight is correct.)

□

I. With high probability the bad weight is small

Observe that, regardless of any other event in the protocol, the probability that a given node of level l is bad is $e^{-\theta(2^l/l^2)}$.

We will argue in each level of the tree separately, showing that with high probability only a small fraction of the nodes in that level are bad. Let M be any constant.

Proposition 1 *With probability $1 - e^{-\theta(N/\log^4 N)}$, at most $N/M2^l l^2$ of the $N/2^l$ nodes at level l are bad.*

Proof: Let $b_l = N/M2^l l^2$, and let $q_l = P(\text{more than } b_l \text{ nodes at level } l \text{ are bad})$.

By a union bound and by the independence in our bound on the probability of any node being bad,

$$q_l \leq \binom{N2^{-l}}{b_l} e^{-\theta(2^l/l^2)b_l}.$$

Now we apply the standard entropy bound $\binom{N}{xN} \leq e^{NH(x)}$ (valid for all $0 < x < 1$; H is the entropy function $H(x) = x \log 1/x + (1-x)\log 1/(1-x)$.) Thus

$$q_l \leq e^{N2^{-l}H(1/Ml^2) - \theta(2^l/l^2)b_l}.$$

Next we apply the inequality $H(x) \leq x^{\frac{\log(e/x)}{\log 2}}$ (valid for all $0 < x < 1$.) Thus

$$\begin{aligned} q_l &\leq e^{N\frac{\log(eMl^2)}{2^l Ml^2 \log 2} - \theta(b_l 2^l/l^2)} \\ &= e^{N\frac{\log(eM)+2\log l}{2^l Ml^2 \log 2} - \theta(N/l^4)} = e^{-\theta(N/l^4)}. \end{aligned}$$

Now since $l \leq \lg N$,

$$q_l \leq e^{-\theta(N/\log^4 N)}.$$

□

Now note that if every level has at most b_l bad nodes, then the total bad weight B of the simulation is at most $\sum_l 2^l b_l = \sum_l N/Ml^2$, which by a suitable choice of M we can arrange to be an arbitrarily small constant fraction of N. Thus, applying the above lemma and taking a union bound over the $\lg N$ levels of the tree (while noting that $\lg N \cdot e^{-\theta(N/\log^4 N)} = e^{-\theta(N/\log^4 N)}$) we find:

Corollary 1 *With probability $1 - e^{-\theta(N/\log^4 N)}$, $B \leq N/34$.* □

II. A simulation with small bad weight is correct

In this section we will apply an amortized analysis to relate the performance of the nodes (as measured by the total bad weight of the simulation) to the progress of the simulation.

At any point in the simulation, and in particular when it terminates, the prefix of agreement of the two processors' *rec* vectors represents the portion of π which has been correctly simulated. Recall that the length of this prefix is equal to the **position** minus the **gap**. Thus the simulation is successful if, at termination, this difference is at least n.

Proposition 2 *The simulation completes at least $N - 17B$ steps of π correctly.*

Recall that the simulation consists of running the protocol of section 4 for N rounds (with $N \geq 2n$). Thus the simulation is successful if $B \leq N/34$.

Proof: We mark every node of the simulation according to its behavior. There are three kinds of nodes:

Bad: As defined earlier: either a faulty simulation round, or a bad check.

GC: Good, correcting: These are good checks in which the processors were in case 2 of the protocol: i.e. they found that the protocol was in error, and corrected the error by bringing their pointers back to a step at which their *rec* vectors are in agreement.

GNC: Good, noncorrecting: These are good rounds; or good checks in which both processors were in either case 1 or case 3 of the protocol. This means that either they found that the simulation is correct, and continued (case 1); or they found that the protocol was in error but could not correct it (case 3).

In terms of these categories, a gap can be created or increased in only two ways in the protocol: (1) A bad node at level l can increase the gap by at most 2^{l+1}. (2) Immediately following a bad check, one or both of the processors may mistakenly advance their pointer one step (as if they were in case 1, although they are not), thus contributing 1 to the gap.

We will charge both of these effects to the bad nodes, thus saying that a bad check can increase the gap by at most $2^{l+1} + 1$, while a bad simulation round can increase the gap by at most 1. Observe that the contribution to the gap is at most 5/2 times the bad weight of the node.

A good node (of either kind) never contributes to the gap. This is true of good leaves (simulation rounds) because they can contribute gap only if they follow immediately after a bad check, to which we have charged the gap. It is true of good internal nodes (checks) because the processors agree on their assessment of the state of the protocol, and therefore either eliminate or avoid increasing the gap; furthermore if there is an error which they cannot correct (case 3), they avoid increasing the gap in the following simulation round.

Observe that after a GC node the gap is always 0 (thus the processors' pointers agree and the simulation is correct through that step).

The length of the final prefix of agreement is equal to the number of correct simulation rounds in the protocol, minus the position loss (from Bad and GC nodes) and any gap remaining at termination. Thus the proposition will follow from the two following claims, which we prove separately:

(A) At least $N - 5B$ of the N simulation rounds of the protocol, are correct simulations of the steps of π.

(B) The total position loss at Bad and GC nodes, plus the gap remaining when the protocol halts, is at most $12B$.

Proof of claim (A). Observe that a simulation round is correct provided that it starts with 0 gap, and that it itself is good.

We will isolate certain subtrees of the simulation, which we will call **dismissable**, and argue that all the simulation rounds outside the dismissable subtrees are correct.

We say that the subtree rooted at a node v of level l is dismissable if:

1. The bad weight in the subtree is at most $\frac{2}{5}2^l$. (In particular v must be good.)
2. One of the subtrees rooted at a child of v, has bad weight more than $\frac{2}{5}2^{l-1}$.
3. No subtree rooted at an ancestor of v satisfies properties 1,2. (Thus dismissable subtrees are disjoint).

The definition implies that every bad node is contained within some dismissable subtree, unless $B > \frac{2}{5}N$ in which case the proposition is trivial. Now we show that all nodes outside of the dismissable subtrees, start with 0 gap. No gap is created outside of the dismissable subtrees, because all the nodes on the outside (as well as the roots of the dismissable subtrees) are good. Therefore it suffices to show that every dismissable subtree ends with 0 gap. By induction (on their order of occurrence in the simulation, namely postorder) we can assume that each dismissable subtree starts with 0 gap. Since the total bad weight inside a dismissable subtree rooted at level l is at most $\frac{2}{5}2^l$, the total gap encountered (if any) by the check node at its root is at most 2^l. Since the root is good, it can correct this, hence the subtree ends with 0 gap.

We now place an upper bound on the number of leaves within the dismissable subtrees. A dismissable subtree rooted at level l has 2^l leaves, and bad weight at least $\frac{2}{5}2^{l-1}$. Since the dismissable subtrees are disjoint, the total number of leaves in their union is at most $5B$.

We have shown that any leaf (simulation round) outside the union of the dismissable subtrees is good, and starts with 0 gap; and furthermore that there are at least $N - 5B$ such leaves. Claim (A) follows.

Proof of claim (B). The only ways in which position is lost in the protocol are:

(i) A bad node at level l can decrease the position by at most 2^{l+1}.

(ii) A GC node at level l can decrease the position by at most 2^{l+1}.

Our argument now runs as follows. The difference between the number of correctly simulated steps of π (which we lower-bounded in (A)), and the length of the final prefix of agreement of the *rec* vectors, is equal to the sum of:

(a) The total position loss as in (i), (ii) above.

(b) The gap remaining at the end of the simulation.

We will amortize each of these effects against the bad weight of the simulation. We have indicated above how gap is amortized against bad weight, at a rate of 5/2 steps per unit of bad weight. Next we examine the position loss, L.

(i) The total position loss at the bad nodes is bounded by $2B$.

(ii) The amortization of position loss at the GC nodes against bad weight must be made indirectly, through an intermediate amortization against the gap being corrected at the GC nodes. The gap is due entirely to the effect of bad nodes (see (1,2) above); and as observed, the total amount of gap to be corrected is bounded by $\frac{5}{2}B$. Thus as an intermediate step, we

will say that a bad node at level l creates $\frac{5}{2}2^l$ "gap coins". Next we show that the GC nodes convert gap coins into position loss, at a constant rate.

Lemma 2 *The total position loss at the GC nodes is at most four times the total gap currency.*

Proof: The gap coins created at any bad node will be used to pay for the the position loss at the next-occurring GC node (i.e. the next in postorder in the simulation tree). Thus a gap coin will be used at a unique GC node. Since a GC node at level l can reduce the position by at most 2^{l+1} steps, the lemma will follow from showing that it has at least 2^{l-1} gap coins available.

We prove this as follows. Recall that immediately after a GC node (as well as at the outset of the simulation), the gap is always 0. Therefore a GC node always receives, in the above scheme, at least as much gap currency as the size of the actual gap which it encounters.

First we consider the level-1 GC nodes. They encounter a gap of at least one step, and therefore have at least one gap coin available; furthermore they reduce the position by at most 4 steps.

Next we consider a GC node v of level $l > 1$. If v's right-hand child is bad, then at least $\frac{5}{2}2^{l-1}$ gap coins flow to v, and we are done. Otherwise the right-hand child is good; and since it left gap for v to correct, it must be a GNC node. Since it could correct any gap of up to 2^{l-1} steps, the actual gap encountered by v — and therefore the gap currency available — is at least 2^{l-1}. □

Write $B = B_1 + B_2$, where B_1 represents the total bad weight occurring before the last GC node of the protocol, and B_2 is the weight due to later bad nodes. The total position loss at Bad nodes of the protocol is at most $2B$; while the above lemma shows that the total position loss at GC nodes is at most $10B_1$. Furthermore the gap remaining at termination is at most $\frac{5}{2}B_2$. Thus the total position loss is at most $(2+10)B_1 + (2+\frac{5}{2})B_2 \leq 12B$. Part (B) of the proposition follows.

This completes the proof of the proposition, and of the public-coin coding theorem. □

6 Private Coins

The proofs of Shannon's coding theorem are via the probabilistic method. In variations, the proofs consist of describing a random coding technique, and showing that most random seeds give codes with favorable qualities; hence there exists a good code.

Our technique is similar. We have described a public-coin random protocol such that on any input pair, most random seeds give a protocol with favorable qualities. Unfortunately in the present case, due to the possibility that different input pairs favor different random seeds, it is not as easy to remove the randomness in favor of an existential statement. Nevertheless we can remove enough randomness to turn our public-coin protocol into a private-coin protocol.

The public-coin protocol used $\theta(n^2)$ random coins, i.e. a sample space of size $2^{\theta(n^2)}$. Theorem 3 showed that for any input pair, if random seeds in this space were sampled uniformly, the average conditional probability of the simulation failing was at most $e^{-n^{1-4\log\log n/\log n}}$.

We will show that there exists a relatively small subset Q of these random seeds, of size $2^{\theta(n)}$, such that for any input, if the random seeds in Q are sampled uniformly, the average conditional probability of the simulation failing is $e^{-n^{1-o(1)}}$.

The processors thus run the private-coin protocol as follows: one processor picks an element of Q uniformly. Then the seed is identified to the other processor with $O(\frac{n}{C})$ channel transmissions (using a Shannon code, and erring with probability $e^{-\theta(n)}$); following which the processors run the public-coin protocol on the selected seed.

Proposition 3 *There exists a set of random seeds Q, of size $q = 2^{\theta(n)}$, such that for every input pair, the protocol which uses a uniformly selected seed from Q, succeeds in simulating π with probability $1 - e^{-\theta(n^{1-4\log\log n/\log n})}$.*

Proof: Omitted. □

This completes the demonstration of the private-coin coding theorem (theorem 2). □

7 Discussion

Insofar as interactive protocols represent the workings of a computer whose inputs and processing power are not localized, we have presented here a coding theorem for computation. There are two questions prompted directly by this theorem. The first is, whether it can be reproduced with a deterministic (rather than randomized private-coin) protocol. The second regards the constant σ: is it 1, as for transmission?

It seems also interesting and important to ask, with regard to networks of more than two processors: to what extent might the coding theorem for interactive protocols be extended to the efficient simulation

of noiseless distributed protocols, on networks with noise.

In this connection we draw attention to the work of Gallager [10], who has previously considered the problem of noise occurring in a distributed computation. He allowed for a complete network of n processors, each of which in a single transmission can broadcast one bit, which arrives at each of the other processors subject to independent noise. He studied a specific problem on this network: supposing that each processor receives a single input bit, he showed how to efficiently and reliably compute the combined parity of all the inputs.

Karchmer and Wigderson [11] observed a certain equivalence between communication complexity and circuit complexity, and thereby stimulated great recent interest in communication complexity. While noisy circuits have been studied in the literature (e.g. [24, 16, 18, 17, 1, 2, 8, 20]), the correspondence between circuits and communication protocols does not appear to extend to the noisy cases of each. Elias [3] and later Peterson and Rabin [15] investigated the possibility of encoding data for computation on noisy gates, and the extent to which these gates might be said to have a finite (nonzero) capacity for computation. (Here as for channel transmission, noiseless encoding and decoding of the data before and after the computation are allowed).

Finally, we mention two lines of research in the literature which, although they address different problems from ours, nevertheless anticipate our work in philosophy or in method. The first is work on sequential coding [25, 19, 4]: a method of implementing data transmission without the computational explosion of block codes. In this method the receiver branches on intermediate guesses regarding the message, and backtracks when inconsistencies mount. Our "progress checks" (section 3) are similarly motivated.

The second is work on the active maintenance of data stored in an unreliable medium (e.g. cellular automata subject to local noise) [22, 6, 7]: a data transmission problem from the past to the future. The use of hierarchical, rather than homogeneous, schemes was found to be important in that work [6]: a theme evident also in the present paper.

Acknowledgments

I thank Mike Sipser, whose encouragement of my work on this project, and supervision of its progress, have been invaluable. Further I thank Peter Elias for much advice and assistance. For consultations and comments thanks also to Dan Abramovich, Robert Gallager, Wayne Goddard, Mauricio Karchmer, Dan Kleitman, Mike Klugerman and Andrew Sutherland.

References

[1] R. L. Dobrushin and S. I. Ortyukov. Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements. Prob. Inf. Trans., 13:59–65, 1977.

[2] R. L. Dobrushin and S. I. Ortyukov. Upper bound for the redundancy of self-correcting arrangements of unreliable functional elements. Prob. Inf. Trans., 13:203–218, 1977.

[3] P. Elias. Computation in the presence of noise. IBM Journal of Research and Development, 2(4):346–353, October 1958.

[4] R. M. Fano. A heuristic discussion of probabilistic decoding. IEEE Transactions on Information Theory, pages 64–74, 1963.

[5] W. Feller. An Introduction to Probability Theory and its Applications, volume I. Wiley, third edition, 1968.

[6] P. Gács. Reliable computation with cellular automata. J. Computer and System Sciences, 32:15–78, 1986.

[7] P. Gács and J. Reif. A simple three-dimensional real-time reliable cellular array. J. Computer and System Sciences, 36:125–147, 1988.

[8] A. Gál. Lower bounds for the complexity of reliable boolean circuits with noisy gates. In Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, pages 594–601, 1991.

[9] R. G. Gallager. Information Theory and Reliable Communication. Wiley, 1968.

[10] R. G. Gallager. Finding parity in a simple broadcast network. IEEE Trans. Inform. Theory, 34(2):176–180, March 1988.

[11] M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In Proceedings of the 20th Annual Symposium on Theory of Computing, pages 539–550, 1988.

[12] Lipton and Sedgewick. Lower bounds for VLSI. In Proceedings of the 13th Annual Symposium on Theory of Computing, pages 300–307, 1981.

[13] L. Lovász. Communication complexity: A survey. In Korde et al, editor, Algorithms and Combinatorics. Springer-Verlag, 1990.

[14] C. H. Papadimitriou and M. Sipser. Communication complexity. In Proceedings of the 14th Annual Symposium on Theory of Computing, pages 196–200, 1982.

[15] W. W. Peterson and M. O. Rabin. On codes for checking logical operations. IBM Journal of Research and Development, 3:163–168, April 1959.

[16] N. Pippenger. On networks of noisy gates. In Proceedings of the 26th Annual Symposium on Foundations of Computer Science, pages 30–36, 1985.

[17] N. Pippenger. Reliable computation by formulas in the presence of noise. IEEE Transactions on Information Theory, 34(2):194–197, March 1988.

[18] N. Pippenger. Invariance of complexity measures for networks with unreliable gates. J. ACM, 36:531–539, 1989.

[19] B. Reiffen. Sequential encoding and decoding for the discrete memoryless channel. Res. Lab. of Electronics, M.I.T. Technical Report, 374, 1960.

[20] R. Reischuk and B. Schmeltz. Reliable computation with noisy circuits and decision trees — a general $n \log n$ lower bound. In Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, pages 602–611, 1991.

[21] C. E. Shannon. A mathematical theory of communication. Bell System Tech. J., 27:379–423; 623–656, 1948.

[22] M. C. Taylor. Reliable information storage in memories designed from unreliable components. Bell System Tech. J., 47(10):2299–2337, 1968.

[23] C. D. Thompson. Area-time complexity for VLSI. In Proceedings of the 11th Annual Symposium on Theory of Computing, pages 81–88, 1979.

[24] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. McCarthy, editors, Automata Studies. Princeton University Press, 1956.

[25] J. M. Wozencraft. Sequential decoding for reliable communications. Res. Lab. of Electronics, M.I.T. Technical Report, 325, 1957.

[26] A. C. Yao. Some complexity questions related to distributive computing. In Proceedings of the 11th Annual Symposium on Theory of Computing, pages 209–213, 1979.

[27] A. C. Yao. The entropic limitations on VLSI computations. In Proceedings of the 13th Annual Symposium on Theory of Computing, pages 308–311, 1981.