Randomness and Computation

ADVANCES IN

COMPUTING RESEARCH

Series Editor: FRANCO P. PREPARATA

Volume Editor: SILVIO MICALI

Volume 5 • 1989

RANDOMNESS AND COMPUTATION

ADVANCES IN
COMPUTING RESEARCH

Volume 5 · 1989

RANDOMNESS AND COMPUTATION

ADVANCES IN COMPUTING RESEARCH

A Research Annual

RANDOMNESS AND COMPUTATION

Series Editor: FRANCO P. PREPARATA
Departments of Electrical and Computer Engineering and of Computer Science
University of Illinois

Volume Editor: SILVIO MICALI
Laboratory for Computer Science
Massachusetts Institute of Technology

VOLUME 5 · 1989

JAI PRESS INC.

Greenwich, Connecticut *London, England*

JAI PRESS LTD.
3 Henrietta Street
London WC2E 8LU
England

ISBN: 0-89232-896-7

Manufactured in the United States of America

CONTENTS

LIST OF CONTRIBUTORS vii

EDITOR'S FOREWORD
Silvio Micali xi

PSEUDORANDOM GENERATORS AND COMPLEXITY CLASSES
Ravi B. Boppana and Rafael Hirschfeld 1

AMPLIFICATION OF PROBABILISTIC BOOLEAN FORMULAS
Ravi B. Boppana 27

PARALLEL TREE CONTRACTION PART 1: FUNDAMENTALS
Gary L. Miller and John H. Reif 47

PRIVATE COINS VERSUS PUBLIC COINS IN INTERACTIVE PROOF SYSTEMS
Shafi Goldwasser and Michael Sipser 73

COLLECTIVE COIN FLIPPING
Michael Ben Or and Nathan Linial 91

ALMOST SORTING IN ONE ROUND
Miklós Ajtai, János Komlós, William Steiger, and Endre Szemerédi 117

CHROMATIC NUMBERS OF RANDOM HYPERGRAPHS AND ASSOCIATED GRAPHS
Eli Shamir 127

ALMOST OPTIMAL LOWER BOUNDS FOR SMALL DEPTH CIRCUITS
John Hastad 143

ANALYSIS OF ERROR CORRECTION BY MAJORITY VOTING
Nicholas Pippenger 171

DETERMINISTIC SIMULATION OF PROBABILISTIC CONSTANT DEPTH CIRCUITS
Miklós Ajtai and Avi Wigderson 199

SELF-CORRECTING TWO-DIMENSIONAL ARRAYS
Peter Gács 223

THE COMPLEXITY OF PERFECT ZERO-KNOWLEDGE
Lance Fortnow 327

RANDOMIZED ROUTING ON FAT-TREES
Ronald L. Greenberg and Charles E. Leiserson 345

FACTORIZATION OF POLYNOMIALS GIVEN BY STRAIGHT-LINE PROGRAMS
Erich Kaltofen 375

A RANDOMIZED DATA STRUCTURE FOR ORDERED SETS
Jon L. Bentley, F. Thomas Leighton, Margaret Lepley, Donald F. Stanat, and J. Michael Steele 413

ON COMPLETENESS AND SOUNDNESS IN INTERACTIVE PROOF SYSTEMS
Martin Furer, Oded Goldreich, Yishay Mansour, Michael Sipser and Stathis Zachos 429

RANDOMIZATION IN BYZANTINE AGREEMENT
Benny Chor and Cynthia Dwork 443

BIASED COINS AND RANDOMIZED ALGORITHMS
N. Alon and M. O. Rabin 499

LIST OF CONTRIBUTORS

Miklós Ajtai
IBM Almaden Research Center
San Jose

N. Alon
Bell Communications Research
Morristown, New Jersey
and
Department of Mathematics
Tel Aviv

Michael Ben Or
Institute of Mathematics and
Computer Science
Hebrew University

Jon L. Bentley
Bell Laboratories
Murray Hill, New Jersey

Ravi B. Boppana
Department of Computer Science
Rutgers University
New Brunswick

Benny Chor
Laboratory for Computer Science
Massachusetts Institute of Technology

Cynthia Dwork
IBM Almaden Research Center
San Jose

Lance Fortnow
Laboratory for Computer Science
Massachusetts Institute of Technology

Martin Furer
Computer Science Department
Pennsylvania State University
University Park

Peter Gács
Department of Computer Science
Boston University

Oded Goldreich
Computer Science Department
TECHNION—Israel Institute of Technology

Shafi Goldwasser
Computer Science Department
Massachusetts Institute of Technology

Ronald I. Greenberg
Laboratory for Computer Science
Massachusetts Institute of Technology

Johan Hastad
Department of Numerical Analysis and Computer Science
Royal Institute of Technology
Sweden

Rafael Hirschfeld
Laboratory for Computer Science
Massachusetts Institute of Technology

Erich Kaltofen
Department of Mathematical Sciences
Rensselaer Polytechnic Institute

János Komlós
Department of Mathematics
University of California at San Diego

F. Thompson Leighton
Mathematics Department and Laboratory for Computer Science
Massachusetts Institute of Technology

Charles E. Leiserson
Laboratory for Computer Science
Massachusetts Institute of Technology

Margaret Lepley
Hughes Aircraft Corporation
Playa Del Rey

Nathan Linial
Institute of Mathematics and Computer Science
Hebrew University

Yishay Mansour
IBM Scientific Center
Haifa

Gary L. Miller
Department of Computer Science
University of Southern California

Nicholas Pippenger — IBM Almaden Research Laboratory, San Jose

M. O. Rabin — Institute of Mathematics, The Hebrew University of Jerusalem, and Division of Applied Sciences, Harvard University

John H. Reif — Computer Science Department, Duke University

Eli Shamir — Institute of Mathematics and Computer Science, Hebrew University

Michael Sipser — Mathematics Department, Massachusetts Institute of Technology

Donald F. Stanat — Department of Computer Science, University of North Carolina, Chapel Hill

J. Michael Steele — Department of Mechnical Engineering, Princeton University

William Steiger — Department of Mathematics, Rutgers University, New Brunswick

Endre Szemerédi — Department of Mathematics, University of California at San Diego

Avi Wigderson — Mathematical Sciences Research Institute, Berkeley

Stathis Zachos — Computer and Information Science, Brookline College of CUNY

FOREWORD

Randomness is successfully contributing to almost all aspects of computation. Even a quick glance at this book shows that it helps in enhancing the performance of some algorithms, in correcting errors occurring during computations, in proving theorems, and so on. This success, perhaps, needs some explanation.

Why does randomness help computation so much? Indeed, randomness and computation are dual players, each standing at opposite ends of the spectrum. Perhaps, the fact that randomness may help computation is no more surprising than how, in Judo, one may use the strength of his opponent to defeat him. Put in another way, the characteristic unpredictability of randomness is a frustration to the rational mind. That is, until we realize that this same unpredictability does not only work against us, but does also frustrate and make ineffective any "adversary" which may arise during a computation. Consider, for instance, an "adversary"—say Nature—

trying to select an input for your algorithm that will oblige it to run for a long time. Her job will be harder if she does not know what your algorithm does, which will indeed be the case if your algorithm flips random coins to decide what to do next.

Why have we realized that randomness can be used in computation? That is, why now? Here is my explanation. I believe the main reason to be that we have become increasingly comfortable in living with uncertainty—the uncertainty, I mean, of human survival. Only after having achieved a good chance of nuclear or ecological disaster could, for instance, the notion of a proof that has a (controllable) chance of being false become so readily acceptable. It is true, to be fair, that nowadays the individual experiences a longer and safer life; but humanity as a whole stands at greater risk. Science is primarily a social enterprise. It is this *common* risk that has paved the way to accepting previously too daring *scientific*—and thus eminently social—concepts. Exciting times indeed, scientifically and otherwise.

I wish to thank all contributors for pausing to meditate in the middle of a revolution, making this book possible. Enjoy it.

Silvio Micali
Volume Editor

PSEUDORANDOM GENERATORS AND COMPLEXITY CLASSES

Ravi B. Boppana and Rafael Hirschfeld

ABSTRACT

This paper explores a notion of computational randomness and its implications for complexity theory. The probabilistic complexity class $\mathcal{BPP}$ is shown to lie within deterministic subexponential time if there exist pseudorandom bit generators whose outputs cannot be distinguished from random strings by polynomial-time statistical tests. This notion of indistinguishability is shown to be equivalent to unpredictability by polynomial-time algorithms. Moreover, certain widely accepted complexity-theoretic assumptions imply the existence of pseudorandom bit generators whose outputs are unpredictable.

Some of the results included here have been published previously without proofs; this paper collects the known results, proves them, and establishes some new related results.

Advances in Computing Research, Volume 5, pages 1–26.

ISBN: 0-89232-896-7

1. INTRODUCTION

An understanding of randomness is of great importance to computer science. Sources of random bits play a key role in cryptography and the design of probabilistic algorithms, and the notion of randomness underlies fundamental theoretical questions concerning probabilistic computation. Though it is difficult to formulate a precise definition that completely captures the intuitive notion of randomness, it is possible to describe some of the expected properties of random sources. Sources that exhibit these properties may be sufficient for specific applications.

True randomness is not likely to be produced by something as inherently deterministic as a computer program. Nonetheless, there are deterministic algorithms, called *pseudorandom generators*, that simulate randomness. A pseudorandom generator takes a short random input, called a *seed*, and expands it into a longer *pseudorandom sequence*. Though the sequences thus produced are not really random, they may appear random in the sense that they satisfy desired properties of truly random strings. Important results can be derived from the existence of a pseudorandom generator whose output is "random enough."

A good pseudorandom generator should be difficult to distinguish from a truly random source. But for any pseudorandom generator there is a statistical test that recognizes its output: given a sequence s, the test consists of an exhaustive search for a seed that generates s. This test is not computationally feasible, however, because it requires more than polynomial time. A pseudorandom generator is "random enough" for many practical and theoretical purposes if its output cannot be distinguished from random strings by statistical tests with polynomially bounded resources. In exciting recent work, Blum and Micali [BM] and Yao [Y] show, under certain assumptions, how to construct such generators.

The recent results about good pseudorandom generators shed new light on the relationship between probabilistic and deterministic complexity classes. Probabilistic complexity classes capture the notion of computation with access to a source of randomness. The ability of a machine to "flip coins" can potentially increase its computational power; there are problems that have efficient probabilistic solutions, but for which no efficient deterministic solutions are known. If good pseudorandom generators exist, however, a deterministic machine can use their outputs to simulate probabilistic computation.

Goldreich, Goldwasser, and Micali [GGM] and Luby and Rackoff [LR] extend the notion of good pseudorandomness to other types of pseudorandom sources. This paper, however, is concerned solely with bit generators.

This paper is a collection of results about pseudorandom generators. Some have been published elsewhere, but many remain unpublished or appear without proofs. They are presented here with a consistent notation and provided with proofs. Some new theorems are also included.

The paper is organized as follows. Section 2 formally defines the notions of pseudorandom generators and statistical tests. Section 3 shows that the definition of pseudorandom generations is robust. Section 4 provides a necessary and sufficient condition for the existence of pseudorandom generators. Section 5 introduces nonuniform generators and compares them to uniform generators. Finally, Section 6 shows that the existence of pseudorandom generators implies that the probabilistic complexity class $\mathcal{BPP}$ is contained in deterministic subexponential time.

2. PSEUDORANDOM COLLECTIONS

One key property of random sequences is *unpredictability*. This property can be formulated in terms of polynomial-time computation by considering the predictive ability of an observer with polynomially bounded resources. Shamir [S] described a pseudorandom number generator for which predicting the next number in the sequence seems to require more than polynomial time. An even stronger source of pseudorandomness is a *bit* generator for which no bit can be feasibly predicted from the preceding bits.[1] Pseudorandom generators unpredictable in this sense were introduced by Blum and Micali [BM]. Informally, a collection of sequences is polynomial-time unpredictable if no polynomial-time algorithm, given the initial segment of a sequence randomly selected from the collection, can predict the next bit of the sequence with probability significantly greater than 1/2. Correctly formulated, this notion of unpredictability is a universal test of randomness with respect to polynomial-time computation. Theorem 2.1, due to Yao [Y], shows that no polynomial-time statistical test can distinguish polynomial-time unpredictable strings from truly random strings of the same length.

Polynomial-time unpredictability could be formulated either with respect to uniform computation or nonuniform computation.

This paper adopts the nonuniform circuit model.[2] It is important to realize, however, that most of the theorems presented here are true with respect to uniform models as well, i.e., they remain true if polynomial-size circuits are replaced by probabilistic polynomial-time Turing machines throughout the paper. Whenever a theorem relies on the nonuniformity of the circuit model, the dependence will be explicitly pointed out.

Pseudorandom sequences are represented as binary strings throughout this paper. The set of all binary strings of length k is denoted by Σ^k. The notions of probability distribution and Boolean circuit used here are the standard ones from probability theory and complexity theory.

DEFINITION. An *ensemble* S is a sequence $\{S_k\}$ such that S_k is a probability distribution on Σ^k. The *random ensemble* $R = \{R_k\}$ is the sequence of uniform distributions, i.e., $R_k(x) = 2^{-k}$ for all $x \in \Sigma^k$.

DEFINITION. A *polynomial-size family of circuits* is a sequence of circuits $C = \{C_k\}$ such that for some constant d, the circuit C_k has at most k inputs and at most k^d nodes.

DEFINITION. Let S be an ensemble. A *next-bit test* is a polynomial-size family of circuits C such that each circuit C_k has $i < k$ inputs and one Boolean output. Let π_k be the probability that on input the first i bits of a sequence s randomly selected from S_k, the output of C_k equals the $(i+1)$st bit of s. The ensemble S *passes* the test C if for all constants d and all sufficiently large k,

$$\pi_k < \frac{1}{2} + \frac{1}{k^d}.$$

DEFINITION. A *c-generator* is a polynomial-time computable function that maps the k-bit strings into the k^c-bit strings, for some integer constant $c > 1$.

Consider the sequences produced by a c-generator G on all seeds of a given length k. They will all be of length k^c, but most sequences of length k^c will not be included, and, since the same pseudorandom sequence may be generated by more than one seed, some may occur more frequently than others. In this way the collection of outputs

of G on inputs of length k defines a probability distribution on Σ^{k^c}. In essence, a c-generator is a means of easily (i.e., in polynomial time) constructing a probability distribution on Σ^{k^c} from the uniform distribution on Σ^k. The ensemble that comprises these distributions is called the ensemble *induced* by G.

The ensembles induced by c-generators are simple ones, because they count occurrences of strings. It is convenient to use probability distributions instead of the collections of strings themselves because complicated combinatorial arguments can then be replaced by simpler techniques from probability theory.

A c-generator is a useful source of pseudorandomness if the ensemble it induces "behaves randomly." This is formalized in terms of unpredictability as follows:

DEFINITION. A c-generator is *strong* if the ensemble it induces passes all next-bit tests.

It might seem that the constant c, which indicates the amount by which a strong c-generator expands its seed, would determine the strength or power of the generator, and that a strong c-generator would be weaker if c were small and stronger if c were large. In fact it will be shown in the next section that the existence of the "weakest" strong generator implies the existence of an arbitrarily "strong" one.

A next-bit test can be viewed as a statistical test. If there is a way to predict the $(i + 1)$st bit of a string chosen from some ensemble S, then it can be used to test whether a given string x is indeed chosen from S: simply apply the predicting circuit to the first i bits of x, and check whether the output matches the $(i + 1)$st bit of x. This will happen with probability $1/2$ if x is truly random, and with probability at least $1/2 + 1/k^d$ if x is selected from S_k.

In general, a statistical test T is an algorithm that on any input produces either a 0 or 1 as output. The test is said to *accept* a string s if it produces a 1 as output when given s as input, and is said to *reject* s otherwise. Suppose that an ensemble S is such that for all but finitely many values of k, the probability that T accepts a string randomly selected from S_k is roughly the same as the probability that T accepts a string randomly selected from R_k.[3] Then T cannot practically distinguish the two distributions; the ensemble S passes the test T. Only after a very large number of experiments can T distinguish between random picks from S_k and random picks from R_k.

Of course, the ensemble induced by a c-generator cannot pass *all* tests. The next theorem will show, however, that the ensemble induced by a strong c-generator will pass all "simple" tests, i.e., those computed by polynomial-size families of circuits.

DEFINITION. If C_k is a circuit with k inputs and 1 output, and S is any ensemble, define $C_k(S)$ to be the probability that C_k accepts a string randomly chosen from S_k.

The probability that a circuit C_k accepts a string randomly chosen from S_k is just the sum of the probabilities of the strings of length k that C_k accepts. Alternatively, it can be thought of as the fraction of strings that C_k accepts in the collection represented by S_k.

DEFINITION. A *polynomial-size statistical test* is a polynomial-size family of circuits. An ensemble S *passes* the test $T = \{T_k\}$ if for every constant d and all sufficiently large k,

$$|T_k(S) - T_k(R)| < \frac{1}{k^d}.$$

We now give a proof of the aforementioned theorem of Yao.

THEOREM 2.1. An ensemble S passes all next-bit tests if and only if it passes all polynomial-size statistical tests.

Proof. A generalization of this theorem is proved by Goldreich, Goldwasser, and Micali [GGM]. The proof included here is similar to the one given by Trilling [T], with some simplifications.

First, some notation is needed. If x is a string of length k, let x_i denote the ith bit of x and let $x_{[i\ldots j]}$ denote bits i through j of x, where $1 \leqslant i \leqslant j \leqslant k$.

Suppose S fails some next-bit test C. Then there is a constant d such that for infinitely many k, the circuit C_k can predict the $(i+1)$st bit of a sequence randomly selected from S_k with probability at least $1/2 + 1/k^d$, for some $i < k$. The circuit family C is used to build a statistical test T as follows: on input x of length k, the test T_k runs C_k on $x_{[1\ldots i]}$. If the output matches x_{i+1}, then T_k outputs a 1; otherwise it outputs a 0. Now if $x \in S_k$, then T_k accepts with probability at least $1/2 + 1/k^d$, but if x is a random k-bit string, then T_k accepts with probability $1/2$. Hence

$$\begin{aligned} T_k(S) - T_k(R) &\geqslant \frac{1}{2} + \frac{1}{k^d} - \frac{1}{2} \\ &= \frac{1}{k^d}, \end{aligned}$$

and so S fails the statistical test T.

Conversely, suppose S fails a statistical test T. Then $|T_k(S) - T_k(R)| \geqslant 1/k^d$, for some constant d and all k in some infinite set K. Choose $k \in K$ and without loss of generality suppose $T_k(S) - T_k(R) \geqslant 1/k^d$. For $0 \leqslant j \leqslant k$, let p_j be the probability that T_k accepts $x_{[1\ldots j]}y$, where x is a string chosen randomly from S_k and y is a string of $k - j$ random bits. Note that $p_0 = T_k(R)$ and $p_k = T_k(S)$. By the pigeonhole principle, there is an i between 0 and $k - 1$ such that

$$p_{i+1} - p_i \geqslant \frac{1}{k \cdot k^d}$$

$$= \frac{1}{k^{d+1}}.$$

The following probabilistic polynomial-time algorithm A can predict the $(i + 1)$st bit: on input $x_{[1\ldots i]}$, the algorithm A flips $k - i$ coins and appends the random bits obtained to its input. It then runs T_k on this extended sequence. If T_k accepts, then A outputs the bit that it randomly chose for the $(i + 1)$st position. If T_k rejects, then A outputs the complement of this bit. The effect of this strategy is to transfer the advantage of the statistical test to the prediction of the $(i + 1)$st bit.

Let q be the probability that T_k accepts $y = x_{[1\ldots i]}\bar{x}_{i+1}r$, where x is randomly chosen from S_k and r is randomly chosen from R_{k-i-1}. In other words, y is the first i bits of a string randomly selected from S_k, followed by the wrong bit, followed by $k - i - 1$ random bits. Notice that

$$p_i = \frac{p_{i+1} + q}{2}.$$

The randomly chosen $(i + 1)$st bit is equally likely to be correct as incorrect. The probability that A is correct is thus one-half the probability that a correct bit is maintained plus one-half the probability that a wrong bit is changed, i.e.,

$$\begin{aligned}
\Pr[A \text{ is correct}] &= \tfrac{1}{2}p_{i+1} + \tfrac{1}{2}(1 - q) \\
&= \tfrac{1}{2}\{p_{i+1} + [1 - (2p_i - p_{i+1})]\} \\
&= \tfrac{1}{2} + p_{i+1} - p_i \\
&\geqslant \tfrac{1}{2} + \frac{1}{k^{d+1}}.
\end{aligned}$$

This probability is over both the i-bit prefixes and the $(k - i)$-bit extensions. For any particular extension z, there is some probability (over just the prefixes) that A is correct when it extends its input with z. The probability (over both prefixes and extensions) that A is correct is just the average of these 2^{k-i} individual probabilities. There must be at least one extension z whose individual probability is greater than or equal to the average. Hence A can be made deterministic if it is given z as well as i as input.

In general, there seems to be no efficient way to compute i or z from k. Here is where the nonuniformity of the circuit model is exploited. Since A clearly runs in time polynomial in k, it can be converted into a polynomial-size circuit family C with the appropriate i and z hardwired into each circuit C_k, for all $k \in K$. This yields a next-bit test that can successfully predict S. Hence S fails some next-bit test. □

This theorem demonstrates the equivalence of a general notion of randomness, based on passing all statistical tests, and a specific one, based on unpredictability. Section 4 will show how to construct unpredictable generators, based on a complexity-theoretic assumption. Because unpredictability implies passing all statistical tests, pseudorandom generators that are random in a very general sense with respect to polynomial-time computation are therefore constructible, under the same assumption.

3. ROBUSTNESS OF STRONG GENERATORS

One application of Theorem 2.1 is to show that the definition of a strong c-generator is robust. Specifically, the existence of a strong c-generator is independent of the particular constant c. This fact was pointed out by Goldreich and Micali [GM].

THEOREM 3.1. Let c_1 and c_2 be integer constants greater than 1. If there exists a strong c_1-generator, then there also exists a strong c_2-generator.

This theorem is a direct consequence of the following lemma about extenders. An extender is a function that captures what is in a sense the weakest notion of a pseudorandom generator. Every pseudorandom generator takes a random seed and extends it into

a longer pseudorandom sequence. An extender outputs a sequence that is just one bit longer than its input.

DEFINITION. An *extender* is a polynomial-time computable function that maps k-bit strings to $(k + 1)$-bit strings.

Notice that an extender induces an ensemble in the same way that a c-generator does. Analogously, an extender is strong if the ensemble it induces passes all next-bit tests.

LEMMA 3.2. For all $c > 1$, there exists a strong extender if and only if there exists a strong c-generator.

Proof. One direction is immediate. Any strong c-generator can be easily converted into a strong extender simply by throwing away all but the first $k + 1$ bits of the output it produces from inputs of length k. This extender is guaranteed to be strong because if it were predictable by some next-bit test, the same next-bit test could predict the same bits of the full output of the c-generator, contradicting the assumption that it is strong.

Conversely, let G be a strong extender, and let c be some constant. Let head x denote x_1 and let tail x denote $x_{[2 \ldots |x|]}$. A strong c-generator G' can be defined inductively as follows:

$$G'(x) = b_1(x)\, b_2(x) \cdots b_{k^c}(x)$$

where

$$a_0(x) = x$$

and for $i > 0$,

$$a_{i+1}(x) = \text{tail } G(a_i(x))$$

$$b_{i+1}(x) = \text{head } G(a_i(x)).$$

What is happening here is that the seed is fed into G, the first bit of the output sequence is plucked off and the remainder is fed back into G. This process is repeated until k^c bits have been stripped away, and these bits form the output of G'. This process could really terminate after $k^c - k$ bits have been generated, using the

remainder at this point, $a_{k^c-k}(x)$, as the last k bits, but going all the way to k^c simplifies the proof.

The generator G' clearly extends a string of length k to one of length k^c, and clearly runs in polynomial time. It remains only to prove that it is strong. The proof is by contradiction; assuming that G' is not strong leads to the conclusion that G is not strong either, which violates the hypothesis.

Let S be the ensemble induced by G and S' be the ensemble induced by G'. Suppose S' fails some statistical test T. Then there exists some constant d such that $|T_{k^c}(S') - T_{k^c}(R)| \geqslant 1/k^{cd}$, for infinitely many k. Pick such a k, and without loss of generality assume that $T_{k^c}(S') \geqslant T_{k^c}(R) + (1/k^{cd})$.

For $0 \leqslant i \leqslant k^c$, define $z_i(x) = yb_1(x)\,b_2(x)\cdots b_{k^c-i}(x)$, where x is a random k-bit string and y is a random i-bit string. Let p_i denote the probability that T accepts $z_i(x)$. Note that $z_0(x) = G'(x)$ and that $z_{k^c}(x)$ is a random k^c-bit string, so $p_0 = T_{k^c}(S')$ and $p_{k^c} = T_{k^c}(R)$. Hence, by the pigeonhole principle, there exists an i such that $p_i - p_{i+1} \geqslant 1/k^{cd+c}$.

Now from the circuit T_{k^c}, a probabilistic algorithm A_k can be constructed that distinguishes the outputs of G of length $k+1$. On input x of length $k+1$, the algorithm A_k flips i coins to generate a random i-bit string y, and then runs T_k on the string

$$w = y \cdot \text{head } x \cdot b_1(\text{tail } x) \cdot b_2(\text{tail } x)\cdots b_{k^c-i-1}(\text{tail } x),$$

accepting if and only if T_k accepts. Notice that if $x = G(a)$, for some $a \in \Sigma^k$, then the part of w that follows y coincides with the first $k^c - i$ bits generated by a. Specifically,

$$\begin{aligned}
\text{head } x &= b_1(a) \\
b_1(\text{tail } x) &= b_2(a) \\
b_2(\text{tail } x) &= b_3(a) \\
&\vdots \\
b_{k^c-i-1}(\text{tail } x) &= b_{k^c-i}(a).
\end{aligned}$$

Hence if x is chosen at random from S_{k+1}, the probability that A_k accepts x is p_i. On the other hand, if x is chosen at random from

R_{k+1}, then head x is just a random bit that blends with y, so the probability that A_k accepts x is p_{i+1}. Since p_i and p_{i+1} differ by at least $1/k^{cd+c}$, the algorithm A_k distinguishes S_{k+1} from R_{k+1}. As in the proof of Theorem 2.1, the probabilistic algorithm A_k can be converted into a deterministic circuit C_k, with the value of i and a single choice of the random bits built in. Such a circuit can be built for infinitely many values of k, and the resulting circuits constitute a polynomial-size statistical test that S fails. Therefore G is not strong, a contradiction. □

Because any strong generator can build any other, strong c-generators will be called simply "strong generators" or "strong bit generators" except in contexts where the particular constant c is important. Strong generators are also called "CSPRB generators" (cryptographically strong pseudorandom bit generators) and "CSB generators" in the literature.

4. CONSTRUCTING STRONG GENERATORS

Randomness and efficient computability are in some sense opposite notions. Intuitively, strings that are hard to recognize are random, and strings that are easy to recognize are not. Theorem 2.1 ensures that if there are strong generators, then their outputs are computationally difficult to recognize. Note that this means that a strong generator is an easily computable function that is difficult to invert. This section establishes the reverse relationship: that if there are easily computable functions that are hard to invert, then they can be used to construct strong generators.

If $\mathcal{P} = \mathcal{NP}$, there can be no strong generators. A nondeterministic machine can distinguish pseudorandom strings from random ones by nondeterministically guessing the seed that generates any given string and then verifying that it in fact does. If this can be done deterministically in polynomial time, then it forms the basis of a polynomial-size statistical test that the generator fails.

The construction of a strong generator appears to require an even stronger assumption than $\mathcal{P} \neq \mathcal{NP}$. Blum and Micali [BM] proposed a set of conditions sufficient for strong bit generation; they also show that their conditions are satisfied if the discrete logarithm problem is intractable. Other researchers [ACG, BCS, BBS, GMT, VV, Y] have shown that these conditions can be met if other specific number-theoretic permutations are one-way.

Blum and Micali's conditions imply the existence of a one-way permutation. Yao [Y] showed conversely that the existence of *any* one-way permutation, suitably formulated, is sufficient for meeting Blum and Micali's conditions, and hence for building a strong generator. Very recently, Levin [L] found the minimal conditions for constructing a strong generator. He shows that the existence of a certain type of one-way function (not necessarily a permutation) is both necessary and sufficient for strong pseudorandom bit generation.

DEFINITION. Consider a length-preserving function f; let $f^{(j)}$ denote its jth iterate. The function f is *one-way on its iterates* if

1. It can be computed in polynomial time, and
2. There is a constant d such that for all polynomial-size families $\{C_k\}$ of circuits with k inputs and k outputs, for all sufficiently large k and all $1 \leqslant j \leqslant k^{d+4}$, we have

$$\Pr[C_k(f^{(j)}(x)) \neq f^{(j-1)}(x)] \geqslant \frac{1}{k^d},$$

where x is chosen uniformly from Σ^k.

Levin's proof uses the concept of isolation explained next. Consider a function b mapping Σ^k into Σ, and a function f mapping Σ^k into Σ^k. The function b is (p, T)-*isolated* from f if every circuit A with k inputs and size at most T satisfies

$$|\Pr[A(f(x)) = b(x)] - 1/2| \leqslant p/2,$$

where x is chosen uniformly from Σ^k. Isolation says that $b(x)$ is somewhat hard to predict given $f(x)$.

Levin proves the following result on the effect of exclusive-or on isolation. Let $\oplus$ denote the exclusive-or operation.

THEOREM A.3. If the functions $b_i(x_i)$ are (p, T)-isolated from $f_i(x_i)$ for all $1 \leqslant i \leqslant n$, then for every $\varepsilon > 0$, the function $b_1(x_1) \oplus b_2(x_2) \oplus \cdots \oplus b_n(x_n)$ is $[p^n + \varepsilon, \varepsilon^2(1-p)^2 T]$-isolated from $f_1(x_1) f_2(x_2) \cdots f_n(x_n)$.

The proof of this theorem is technical and is given in the appendix. Levin's result on the existence of strong generators will now be proved.

THEOREM 4.1. There is a function that is one-way on its iterates if and only if there is a strong generator.

Proof. ($\Leftarrow$) Suppose there is a strong c-generator G, with induced ensemble S. Consider the function f mapping k^c-bit strings to k^c-bit strings defined by

$$f(x) = G(x_{[1\ldots k]}).$$

We will show that f is one-way on its iterates, with $d = 1$. Suppose that $1 \leqslant j \leqslant k^5$. Let $\{C_k\}$ be a polynomial-size family of circuits with k^c inputs and k^c outputs.

Consider the statistical test T_j that accepts an input x if and only if $C_k(f^{(j)}(x))$ is not equal to $f^{(j-1)}(x)$. Since S passes all statistical tests, for every constant d and all sufficiently large k, we have

$$|T_j(S) - T_j(R)| < \frac{1}{k^d}.$$

The definition of f implies that $T_j(S) = T_{j+1}(R)$. Thus, by summing over j, for every constant d and all sufficiently large k, we have

$$|T_j(R) - T_1(R)| < \frac{1}{k^{d-5}}.$$

We now show that $T_1(R)$ is large. The value of $C_k(f(x))$ depends only on the first k bits of x. Thus, for a given k-bit prefix, at most one $(k^c - k)$-bit extension will satisfy $C_k(f(x)) = x$. Thus $T_1(R)$ is at least $1 - 2^{k-k^c}$. Combining this result with the last equation of the previous paragraph shows that for every constant d and all sufficiently large k, we have

$$\begin{aligned} T_j(R) &> 1 - 2^{k-k^c} - \frac{1}{k^{d-5}} \\ &\geqslant \frac{1}{k}. \end{aligned}$$

Hence the function f is one-way on its iterates.

($\Rightarrow$) Suppose that the function f is one-way on its iterates. Let d be as promised in the definition of one-way on iterates.

Assume that $k = 2^l$ for some l; the proof can be easily modified if this is not true. Let ρ be any natural map from Σ^l onto the integers $\{1, 2, \ldots, k\}$. For $x \in \Sigma^{k+l}$, let

$$b_{j,k}(x) = \text{the } \rho(x_{[k+1 \ldots k+l]})\text{-th bit of } f_k^{(j-1)}(x_{[1 \ldots k]})$$

$$g_{j,k}(x) = f^{(j)}(x_{[1 \ldots k]})\, x_{[k+1 \ldots k+l]}.$$

Because the function f is one-way on its iterates, the function $b_{j,k}$ is $(1 - 2/k^{d+1}, k^e)$-isolated from $g_{j,k}$, for every constant e and all sufficiently large k.

We now amplify the isolation. Write $y = x_1 x_2 \cdots x_{k^{d+2}}$, where each x_i is in Σ^{k+l}. Define

$$c_{j,k}(y) = \bigoplus_{i=1}^{k^{d+2}} b_{jk}(x)$$

$$h_{j,k}(y) = g_{j,k}(x_1) g_{j,k}(x_2) \cdots g_{j,k}(x_{k^{d+2}}).$$

Applying Theorem A.3 shows that $c_{j,k}$ is $(e^{-k} + \varepsilon, 4\varepsilon^2 k^{e-2d-2})$-isolated from $h_{j,k}$, for every constant e, all sufficiently large k, and all $\varepsilon > 0$.

Consider the bit generator G mapping strings of length $k^{d+2}(k + l)$ to strings of length k^{d+4} defined by

$$G(y) = c_{k^{d+4},k}(y) \cdots c_{2,k}(y)\, c_{1,k}(y).$$

Because f is computable in polynomial time, so is G. It remains to show that G is strong. A next-bit test is of the form: given $c_{k^{d+4},k}(y) \cdots c_{j+1,k}(y)$, try to predict $c_{j,k}(y)$. Since $c_{k^{d+4},k}(y), \ldots, c_{j+1,k}(y)$ can be computed from $h_{j,k}(y)$ in polynomial time, it suffices to show that predicting $c_{j,k}(y)$ from $h_{j,k}(y)$ is difficult. But this is precisely what the isolation obtained above shows. □

5. UNIFORM AND NONUNIFORM GENERATORS

A strong generator is a uniform algorithm that expands a random seed into a longer pseudorandom sequence. But consider instead a family of circuits that does the same thing, with each circuit working for a different input length. Such a polynomial-size

family of circuits is called a *nonuniform strong generator*. Even if the circuit family is polynomial-size, it does not necessarily constitute a strong generator, for it may be very inefficient (or even impossible) for any algorithm to construct the appropriate circuits.

Polynomial-size families of circuits compute the same functions as deterministic polynomial-time algorithms with polynomial-length *advice*, where the same advice string is used for all inputs of a given length. (A circuit family can be simulated by a universal algorithm that is given a description of the corresponding circuit as advice; an algorithm with advice can be simulated by a circuit family by hardwiring the advice strings into the circuits.) This yields an alternate definition of a nonuniform generator as a polynomial-time algorithm with polynomial-length advice.

THEOREM 5.1. Let c_1 and c_2 be integer constants greater than 1. If there exists a nonuniform strong c_1-generator with advice of length k^d, then there also exists a nonuniform strong c_2-generator with advice of length k^d.

Proof. In the proof of Lemma 3.2, the outputs of the extender were fed back into the extender in order to lengthen the sequence produced. The same technique works for nonuniform extenders. Since the extender is always given strings of the same length, the same advice can be used each time. The resulting generator still runs in polynomial time with polynomial-size advice. Hence if a nonuniform extender exists, then so does a nonuniform c-generator, for all $c > 1$. The theorem follows. □

As a potential example of uniform versus nonuniform generators, consider the discrete logarithm problem, upon which the first constructions (depending on complexity-theoretic assumptions) of strong generators were based.

DEFINITION. Let p be a prime. The set of integers $\{1, \ldots, p-1\}$ forms a cyclic group, denoted $\mathbf{Z}_p^*$, under multiplication modulo p. Given a prime p, a generator g for $\mathbf{Z}_p^*$, and an element y of $\mathbf{Z}_p^*$, the *discrete logarithm problem* (DLP) is to find the unique element x of $\mathbf{Z}_p^*$ such that $y = g^x \bmod p$.

The discrete logarithm problem seems to be computationally difficult. Although it is not known to require superpolynomial

time, the best known algorithm, due to Adleman [A], takes time $2^{\Theta(\sqrt{\log p \log\log p})}$. On the other hand, its inverse, the discrete exponential, can be computed in polynomial time (by repeated squaring), and is hence a good candidate for a one-way function.

Two different intractability assumptions have been proposed for the DLP, one by Blum and Micali [BM] and the other by Yao [Y].

ASSUMPTION 1. (Blum, Micali) Let $C = \{C_k\}$ be any polynomial-size family of circuits. Then C computes the discrete logarithm problem for less than a fraction $1/k^c$ of the primes of length k for all c and all sufficiently large k.

ASSUMPTION 2. (Yao) Let $l(p)$ be the size of the smallest circuit that solves the DLP for the prime p, and let $L(k)$ be the maximum value of $l(p)$ for all primes of length at most k. Then L grows asymptotically faster in k than any polynomial.

Blum and Micali [BM] prove that Assumption 1 is sufficient for the construction of a strong generator. Assumption 2, though sufficient to demonstrate the existence of a nonuniform strong generator, appears inadequate for the construction of a uniform one. Some consequences of the existence of uniform and nonuniform strong generators are discussed in the next section.

6. STRONG GENERATORS AND COMPLEXITY CLASSES

The preceding sections defined strong pseudorandom generators and showed how to construct them from a one-way function; this section will show how to use strong pseudorandom generators to simulate probabilistic computation deterministically. Specifically, the successive bits output by a generator are used to determine the outcome of successive coin flips in the computation of a probabilistic machine. The theorems in this section use this simulation technique to relate deterministic and probabilistic complexity classes.

DEFINITION. A *probabilistic Turing machine* is a special case of a nondeterministic Turing machine in which every step of the computation has either one or two successors. In the case of two successors, the next step is determined by flipping a coin. Without loss of generality, on any input all branches of the computation are assumed to have the same number of coin flips. The probability that

a machine M accepts an input w, denoted Pr[M accepts w], is the number of accepting branches divided by the total number of branches.

A probabilistic Turing machine that runs in time $t(n)$ can be thought of as a deterministic machine with a special one-way read-only tape of length $t(n)$. The machine reads a new bit from this tape whenever it wants to flip a coin. The probability of accepting an input w of length n is the fraction of all $t(n)$-bit strings that cause the machine to accept when started with that string on its special tape and w on its input tape.

DEFINITION. The probabilistic complexity classes $\mathcal{R}$ and $\mathcal{BPP}$ are defined as follows (here M refers to a polynomial-time probabilistic Turing machine):

$$\mathcal{R} = \left\{ L \mid \exists M \begin{array}{l} w \in L \Rightarrow \Pr[M \text{ accepts } w] \geqslant \frac{1}{2} \\ w \notin L \Rightarrow \Pr[M \text{ accepts } w] = 0 \end{array} \right\}$$

$$\mathcal{BPP} = \left\{ L \mid \exists M \begin{array}{l} w \in L \Rightarrow \Pr[M \text{ accepts } w] \geqslant \frac{3}{4} \\ w \notin L \Rightarrow \Pr[M \text{ accepts } w] \leqslant \frac{1}{4} \end{array} \right\}$$

It is easy to see that $\mathcal{R} \subseteq \mathcal{BPP}$. Let $L \in \mathcal{R}$ be accepted by some machine M. A machine M' simulates M with input w by running it twice and accepting if either run accepts. If $w \notin L$, the probability of acceptance remains 0. If $w \in L$, the probability of rejection is squared, so the probability of acceptance is boosted to at least 3/4. Hence $L \in \mathcal{BPP}$.

Yao shows that under his assumption for the discrete logarithm problem, the class $\mathcal{R}$ is contained within deterministic subexponential time. The proof uses the output of DLP-based generators to simulate coin tosses. Intuitively, a polynomial-time indistinguishable generator should work equally well for any reasonable polynomial-time probabilistic complexity class, so one might expect the result to hold for $\mathcal{BPP}$ as well. This is of especial interest because some important problems in $\mathcal{BPP}$ are not known to be in $\mathcal{R}$ [BMS].

Although it seems difficult to extend Yao's proof to $\mathcal{BPP}$, the stronger result does follow from Blum and Micali's intractability assumption. In fact, the following theorem shows that a strong generator can be used to recognize $\mathcal{BPP}$ languages in deterministic subexponential time. This establishes the result in its full generality,

since it depends not on the DLP but on the existence of any strong generator, of which DLP-based generators are only one instance. In particular, the generators produced from one-way functions in Levin's theorem satisfy the hypothesis. The proof distills the essential ideas of Yao's proof.

THEOREM 6.1. If there exists a strong generator, then

$$\mathcal{BPP} \subseteq \bigcap_{\varepsilon>0} \mathrm{DTIME}(2^{n^\varepsilon}).$$

Proof. Choose an integer $c > 1$ and let $\varepsilon = 1/c$. Let $L \in \mathcal{BPP}$ be accepted by some probabilistic machine M_L in time n^j, where n is the length of the input. By Theorem 3.1, there is a strong (cj)-generator G that runs in time k^d, for some d (here k is the length of the seed). A deterministic machine M can simulate M_L by using pseudorandom bits generated by G instead of coin flips to determine the computation path. The machine M repeats the simulation for all of the n^j-bit pseudorandom sequences, i.e., for all of the n^ε-bit seeds, and accepts if and only if more than half of them lead to accepting computations of M_L. This takes time

$$(2^{n^\varepsilon})(n^{\varepsilon d})(n^j),$$

i.e., the number of seeds, times the time to expand a seed into a pseudorandom sequence, times the time to simulate M_L on a particular sequence. The simulation may entail some overhead, but this is negligible.

Suppose $L(M)$ and L are unequal. If they differ on only finitely many strings, then M can be patched without increasing its asymptotic running time. Otherwise, for infinitely many k there is a string y_k of length k on which they differ. For each y_k there is a circuit C_k with k^j inputs and size k^b (for some constant b independent of k) that simulates M_L on y_k using its inputs to determine the probabilistic branches. The strings y_k may not be easy to determine, so it is necessary to take advantage of nonuniformity and build them into the circuits. Let $C = \{C_k\}$. Then C is a polynomial-size statistical test that G fails, since, for arbitrarily large k, either

$$\left.\begin{aligned} C_k(R) &> \tfrac{3}{4} \\ C_k(S) &< \tfrac{1}{2} \end{aligned}\right\} \text{(if } y_k \in L\text{)}, \qquad \text{or} \qquad \left.\begin{aligned} C_k(R) &< \tfrac{1}{4} \\ C_k(S) &> \tfrac{1}{2} \end{aligned}\right\} \text{(if } y_k \notin L\text{)}.$$

In either case,

$$|C_k(R) - C_k(S)| > \tfrac{1}{4}.$$

Hence $L(M) = L$, and $L \in \mathrm{DTIME}(2^{n^\varepsilon} n^{\varepsilon d + j}) \subseteq \mathrm{DTIME}(2^{n^{2\varepsilon}})$. Because c can be chosen arbitrarily large, ε can be made arbitrarily small, and therefore $L \in \bigcap_{\varepsilon > 0} \mathrm{DTIME}(2^{n^\varepsilon})$. □

A nonuniform generator yields the following weaker result:

THEOREM 6.2. If there exists a nonuniform strong generator, then

$$\mathscr{R} \subseteq \bigcap_{\varepsilon > 0} \mathrm{DTIME}(2^{n^\varepsilon}).$$

Proof. Choose an integer $c > 1$ and let $\varepsilon = 1/c$. Let $L \in \mathscr{R}$ be accepted by some probabilistic machine M_L in time n^j, where n is the length of the input. The nonuniform strong generator can be computed by a polynomial-time algorithm with advice of length k^d, for some d. By Theorem 5.1, there is for every $b > 1$ a nonuniform strong b-generator that can also be computed by an algorithm with the same advice. The deterministic simulation of M_L is similar to that of the previous theorem except that since the good advice is not known, all advice strings of the appropriate length are tried. Since there are 2^{k^d} advice strings, it is necessary only to choose b large enough so that seeds of length $n^{\varepsilon/d}$ are sufficient to generate sequences of length n^j, i.e., choose $b > cdj$. The simulating machine accepts if it finds *any* accepting computation.

The rest of the proof is the same as before except that the probabilities of acceptance are changed to fit the definition of $\mathscr{R}$. □

It is not known which circuit of a given size is part of a nonuniform strong generator, but only that at least one of them is, so all of them are tried. Since it is necessary only to find a single accepting path of the computation, a bad generator cannot hurt even if it generates only rejecting paths. The one-sided error of $\mathscr{R}$ ensures that bad generators do not affect the outcome.

By contrast, the existence of a nonuniform strong generator seems insufficient to show deterministic containment for the more

general class $\mathscr{BPP}$. Because $\mathscr{BPP}$ is defined to have two-sided error, a simulating machine much check that more than half of the paths accept. On inputs that should be accepted, the spurious generators may generate too many rejecting paths, and on inputs that should be rejected, they may generate too many accepting paths. The crucial difference is that, unlike for $\mathscr{R}$, a single witness does not suffice; the results must be weighed, and the bad generators can disrupt the balance.

APPENDIX: THE ISOLATION THEOREM

This appendix shows how to amplify functions that are somewhat hard to compute to obtain new functions that are almost everywhere difficult to compute. The amplification is accomplished by taking the exclusive-or of independent copies of the original function. This technique was first developed by Yao [Y]; see also Goldwasser [G]. Levin [L] proves that exclusive-or is an even better security amplifier than previously known.

Given a function b from Σ^k to Σ, say that b is (p, T)-*isolated* if every circuit A with k inputs and size at most T satisfies $|\Pr[A(x) = b(x)] - 1/2| \leqslant p/2$, where x is chosen uniformly from Σ^k. The following lemma due to Levin shows that taking the exclusive-or of two functions essentially multiplies the isolation.

LEMMA A.1. If the function $b_1(x_1)$ is (p_1, T_1)-isolated and the function $b_2(x_2)$ is (p_2, T_2)-isolated, then for every $\varepsilon > 0$, the function $b_1(x_1) \oplus b_2(x_2)$ is $(p_1p_2 + \varepsilon, T)$-isolated, where $T = \min(T_1, \varepsilon^2 T_2)$.

Proof. Suppose that the function $b_1(x_1)$ is (p_1, T_1)-isolated and that the function $b_2(x_2)$ is (p_2, T_2)-isolated. Given $\varepsilon > 0$, set $T = \min(T_1, \varepsilon^2 T_2)$. Consider a circuit $A(x_1, x_2)$ of size at most T. We will show that A can predict $b_1(x_1) \oplus b_2(x_2)$ with advantage no better than $p_1p_2 + \varepsilon$. For a fixed x_2, view $A(x_1, x_2) \oplus b_2(x_2)$ as a predictor for $b_1(x_1)$. Since $b_1(x_1)$ is (p_1, T_1)-isolated, every x_2 satisfies

$$|\Pr[A(x_1, x_2) = b_1(x_1) \oplus b_2(x_2)] - 1/2| \leqslant p_1/2,$$

where x_1 is chosen uniformly from the domain of b_1.

Consider the following algorithm $L(x_2)$ for predicting $b_2(x_2)$. The values of $y_1, y_2, \ldots, y_n$ are chosen uniformly from the domain

of b_1, and the real number t is chosen uniformly from the interval $[c, d]$, where $c = (1 - p_1)n/2$ and $d = (1 + p_1)n/2$.

Algorithm $L(x_2)$
  $sum \leftarrow |\{1 \leqslant i \leqslant n\colon A(y_i, x_2) \oplus b_1(y_i) = 1\}|$
  if $sum \geqslant t$
    then $L(x_2) \leftarrow 1$
    else $L(x_2) \leftarrow 0$
end algorithm

The algorithm L can be implemented as a circuit of size $nT + O(n)$. How well does $L(x_2)$ predict $b_2(x_2)$? We have

$$\begin{aligned}\Pr[L(x_2) = b_2(x_2)] &= \Pr[b_2(x_2) = 1 \wedge sum \geqslant t] \\ &\quad + \Pr[b_2(x_2) = 0 \wedge sum < t] \\ &= \Pr[b_2(x_2) = 1 \wedge sum \geqslant t] \\ &\quad + \Pr[b_2(x_2) = 0 \wedge n - sum > n - t].\end{aligned}$$

Since t is symmetric about $n/2$, we obtain

$$\Pr[L(x_2) = b(x_2)] = \Pr[s \geqslant t],$$

where $s = |\{1 \leqslant i \leqslant n\colon A(y_i, x_2) = b_1(y_i) \oplus b_2(x_2)\}|$.

Divide the possible values of s into three intervals: $s \leqslant c$, $s \geqslant d$, and $c < s < d$. Given a statement r, let $\delta(r)$ equal 1 if r is true, and equal 0 otherwise. Write $E[\cdot]$ for expected value. Splitting into the three intervals yields

$$\begin{aligned}\Pr[s \geqslant t] &= E[(1) \cdot \delta(s \geqslant d)] \\ &\quad + E\left[\left(\frac{s - c}{p_1 n}\right) \cdot \delta(c < s < d)\right] \\ &\quad + E[(0) \cdot \delta(s \leqslant c)],\end{aligned}$$

which upon rearranging gives

$$\begin{aligned}\Pr[s \geqslant t] &= E\left[\frac{s - c}{p_1 n}\right] \\ &\quad + E\left[\left(1 - \frac{s - c}{p_1 n}\right) \cdot \delta(s \geqslant d)\right] \\ &\quad + E\left[-\left(\frac{s - c}{p_1 n}\right) \cdot \delta(s \leqslant c)\right].\end{aligned}$$

Let I_1, I_2, and I_3 denote the first, second, and third terms, respectively, of the right-hand side of this equation. We can write I_1 as follows:

$$\begin{aligned} I_1 &= \frac{1}{p_1 n} E[s] - \frac{1 - p_1}{2p_1} \\ &= \frac{1}{p_1} \Pr[A(x_1, x_2) = b_1(x_1) \oplus b_2(x_2)] - \frac{1}{2p_1} + \frac{1}{2} \\ &= \frac{1}{2} + \frac{1}{p_1}(\Pr[A(x_1, x_2) = b_1(x_1) \oplus b_2(x_2)] - \tfrac{1}{2}), \end{aligned}$$

where x_1 and x_2 are chosen uniformly from the domains of b_1 and b_2, respectively.

We now show that the terms I_2 and I_3 are small in absolute value. Rearranging I_2 shows that

$$I_2 = \frac{-1}{p_1 n} E[(s - d)\,\delta(s \geqslant d)],$$

where the expectation is taken over random x_2 and random $y_1, y_2, \ldots, y_n$. Rearranging I_3 gives a similar expression, so by symmetry it suffices to estimate I_2.

Consider a fixed value of x_2. Let p equal $\Pr[A(x_1, x_2) = b_1(x_1) \oplus b_2(x_2)]$, where the probability is taken over x_1. The beginning of the proof showed that p is between $(1 - p_1)/2$ and $(1 + p_1)/2$. The random variable s is the sum of n independent Bernoulli trials, each trial having probability p of success, i.e., the random variable s has a binomial distribution. We are trying to estimate

$$E[(s - d)\,\delta(s \geqslant d)].$$

This expectation is maximized when p is as large as possible, so to bound the expectation from above, we shall assume that p equals $(1 + p_1)/2$. The expression $E[(s - pn)\,\delta(s \geqslant pn)]$ can be evaluated in closed form; backward induction on k shows that

$$E[(s - pn)\,\delta(s \geqslant k)] = n\binom{n-1}{k-1} p^k (1 - p)^{n+1-k}.$$

Estimating the right-hand side by Stirling's formula yields

$$E[(s - pn)\,\delta(s \geqslant pn)] \leqslant \sqrt{\frac{p(1-p)\,n}{2\pi}} = \sqrt{\frac{(1-p_1^2)\,n}{8\pi}}.$$

This bound was proved for every fixed x_2, so by averaging, the bound remains true for a random x_2.

The preceding analysis shows that $|I_2|$ is bounded by

$$|I_2| \leqslant \frac{1}{p_1 n}\sqrt{\frac{(1-p_1^2)\,n}{8\pi}} = \frac{1}{p_1}\sqrt{\frac{1-p_1^2}{8\pi n}}.$$

The term $|I_3|$ has the same upper bound. Since $I_2 \leqslant 0$ and $I_3 \geqslant 0$, the expression $|I_2 + I_3|$ is also bounded by this quantity. Recall the equation

$$\Pr[L(x_2) = b_2(x_2)] = I_1 + I_2 + I_3,$$

From the above remarks, we have

$$|I_1 - 1/2| \leqslant |\Pr[L(x_2) = b_2(x_2)] - 1/2| + \frac{1}{p_1}\sqrt{\frac{1-p_1^2}{8\pi n}}.$$

Since $b_2(x_2)$ is (p_2, T_2)-isolated, we have

$$|\Pr[L(x_2) = b_2(x_2)] - 1/2| \leqslant \tfrac{1}{2}p_2,$$

provided that $nT + O(n) \leqslant T_2$. Combining the last two inequalities yields

$$|I_1 - 1/2| \leqslant \tfrac{1}{2}p_2 + \frac{1}{p_1}\sqrt{\frac{1-p_1^2}{8\pi n}}.$$

Using our previous expression for I_1, we obtain

$$\begin{aligned}|\Pr[A(x_1, x_2) = b_1(x_1) \oplus b_2(x_2)] - 1/2| &\leqslant p_1 |I_1 - 1/2| \\ &\leqslant \frac{1}{2}\left(p_1 p_2 + \sqrt{\frac{1-p_1^2}{2\pi n}}\right),\end{aligned}$$

where x_1 and x_2 are uniform over the domains of b_1 and b_2, respectively. By choosing $n = 1/\varepsilon^2$, the advantage becomes less than $p_1p_2 + \varepsilon$. Since this choice of n satisfies $nT + O(n) \leqslant T_2$, we are done. □

Actually, for the proof of Levin's theorem on strong generators, a more general notion of isolation is needed. Consider a function b mapping Σ^k into Σ, and a function f mapping Σ^k to Σ^k. Say that b is (p, T)-*isolated* from f if every circuit A with n inputs and size at most T satisfies $|\Pr[A(f(x)) = b(x)] - 1/2| \leqslant p/2$, where x is chosen uniformly from Σ^k. Thus $b(x)$ is hard to predict given $f(x)$. The following result is for the more general isolation.

LEMMA A.2. If the function $b_1(x_1)$ is (p_1, T_1)-isolated from $f_1(x_1)$ and the function $b_2(x_2)$ is (p_2, T_2)-isolated from $f_2(x_2)$, then for every $\varepsilon > 0$ the function $b_1(x_1) \oplus b_2(x_2)$ is $(p_1p_2 + \varepsilon, T)$-isolated from $f_1(x_1)f_2(x_2)$, where $T = \min(T_1, \varepsilon^2 T_2)$.

Proof. The proof is almost word-for-word the same as that of Lemma A.1. The only modification required is to replace $A(x_1, x_2)$ with $A(f_1(x_1), f_2(x_2))$. □

We can now prove Levin's isolation theorem, which shows that taking the exclusive-or of many functions dramatically increases the isolation.

THEOREM A.3. If the function $b_i(x_i)$ is (p, T)-isolated from $f_i(x_i)$ for all i between 1 and n, then for every $\varepsilon > 0$ the function $b_1(x_1) \oplus b_2(x_2) \oplus \cdots \oplus b_n(x_n)$ is $[p^n + \varepsilon, \varepsilon^2(1-p)^2 T]$-isolated from $f_1(x_1)f_2(x_2)\cdots f_n(x_n)$.

Proof. Using Lemma A.2, an induction on n will show that the function $b_1(x_1) \oplus b_2(x_2) \oplus \cdots \oplus b_n(x_n)$ is $(q, \varepsilon^2 T)$-isolated from $f_1(x_1)f_2(x_2)\cdots f_n(x_n)$, where q equals $p^n + \varepsilon \cdot (1 + p + p^2 + \cdots + p^{n-2})$. In the induction, apply Lemma A.2 with $\bigoplus_{i=1}^{n-1} b_i(x_i)$ and $b_n(x_n)$ replacing $b_1(x_1)$ and $b_2(x_2)$, respectively. Summing the geometric series for q gives $q < p^n + \varepsilon/(1-p)$. The result follows by replacing ε with $\varepsilon/(1-p)$. □

ACKNOWLEDGMENTS

We would like to thank Silvio Micali, who served as supervisor for the master's thesis on which this paper is based, for invaluable discussions.

This research was supported in part by two NSF graduate fellowships. A preliminary version appeared as an M.I.T. master's thesis [H].

NOTES

1. Blum and Micali [BM] point out that the individual bits of a number may be predictable even if the number itself is unpredictable.

2. A *circuit* is a directed acyclic graph. Nodes with indegree 0 are called *inputs* and nodes with outdegree 0 are called *outputs*. Every node that is not an input is labeled with either $\wedge$ or $\vee$. A circuit with $2j$ inputs and k outputs computes a function from Σ^j to Σ^k in a natural way: given $x \in \Sigma^j$, the inputs are given values equal to the bits of x and their complements, each $\wedge$ gate is given a value equal to the Boolean product of the values of the nodes directed into it, and each $\vee$ gate is given a value equal to the Boolean sum of the nodes directed into it. The values of the outputs constitute the value of the function, which is well defined because the circuit has no cycles.

3. There is a suggestive, thought not entirely accurate, way to visualize this criterion. Picture the pseudorandom sequences of length k produced by a generator as points scattered throughout the set of all strings of length k. Now a statistical test is a line that partitions the entire set into two categories, those strings that it accepts and those that it rejects. This line also partitions the pseudorandom strings. Passing the test means that the pseudorandom points are divided in roughly the same proportion as the entire set. The inaccuracy here is that the same pseudorandom point may occur more than once, but the picture is otherwise correct.

REFERENCES

[A] L. Adleman, "A subexponential algorithm for the discrete logarithm problem with applications to cryptography," *Proc. 20th Annu. IEEE Symp. Found. Computer Sci.* 55–60 (1979).

[ACG] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, "RSA/Rabin bits are $\frac{1}{2} + 1/poly(\log n)$ secure," *Proc. 25th Annu. IEEE Symp. Found. Computer Sci.* 499–457 (1984). To appear in *SIAM J. Computing*.

[BMS] E. Bach, G. Miller, and J. Shallit, "Sums of divisors, perfect numbers, and factoring," *SIAM J. Computing* 15(4) 1143–1154 (November 1986).

[BCS] M. Ben-Or, B. Chor, and A. Shamir, "On the cryptographic security of single RSA bits," *Proc. Fifteenth Annu. ACM Symp. Theory Computing* 421–430 (1983).

[BBS] L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudo-random number generator," *SIAM J. Computing* 15(2) 364–383 (May 1986).

[BM] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. Computing* 13(4) 850–864 (November 1984).

[GGM] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM* 33(4) 792–807 (October 1986).

[GM] O. Goldreich and S. Micali, personal communication.

[G] S. Goldwasser, *Probabilistic Encryption: Theory and Applications*, Ph.D. Dissertation, University of California at Berkeley, 1984.

[GMT] S. Goldwasser, S. Micali, and P. Tong, "Why and how to establish a private code on a public network," *Proc. 23rd Annu. IEEE Symp. Found. Computer Sci.* 134–144 (1982).

[H] R. Hirschfeld, *Pseudorandom Generators and Complexity Classes*, S.M. Thesis, Massachusetts Institute of Technology, 1986.

[L] L. Levin, "One-way functions and pseudorandom generators," *Proc. Seventeenth Annu. ACM Symp. Theory Computing* 363–365 (1985). To appear in *Combinatorica*.

[LR] M. Luby and C. Rackoff, "How to construct pseudo-random permutations from pseudo-random functions," *Proc. Eighteenth Annu. ACM Symp. Theory Computing* 356–363 (1986).

[S] A. Shamir, "On the generation of cryptographically strong pseudorandom sequences," *ACM Trans. Computer Syst.* 1(1), 38–44 (February 1983).

[T] S. Trilling, *Some Implications of Complexity Theory on Pseudo-Random Bit Generation*, S.M. Thesis, Massachusetts Institute of Technology, 1985.

[VV] U. Vazirani and V. Vazirani, "Efficient and secure pseudo-random number generation," *Proc. 25th Annu. IEEE Symp. Found. Computer Sci.* 458–463 (1984).

[Y] A. C. Yao, "Theory and applications of trapdoor functions," *Proc. 23rd Annu. IEEE Symp. Found. Computer Sci.* 80–91 (1982).

AMPLIFICATION OF PROBABILISTIC BOOLEAN FORMULAS

Ravi B. Boppana[1]

ABSTRACT

Amplification is a method of reducing the error probability of a probabilistic Boolean formula, by combining independent copies of the formula. Valiant used the amplification method to prove the existence of monotone Boolean formulas of size $O(n^{5.3})$ that compute the majority function of n variables. In this paper, we prove that the amount of amplification that Valiant obtained is optimal, thus showing a limit to the power of the amplification method. In addition, using the amplification method, we show the existence of monotone Boolean formulas of size $O(k^{4.3} n \log n)$ computing the kth threshold function of n variables.

Advances in Computing Research, Volume 5, pages 27–45.

ISBN: 0-89232-896-7

1. INTRODUCTION

Many researchers have used the following method, which I call the *amplification method*, to show that particular Boolean functions have small Boolean circuits: start with probabilistic Boolean circuits that approximately compute the function; combine independent copies of such circuits to form probabilistic circuits with exponentially small probability of error; by a probabilistic argument, show the existence of small deterministic circuits that exactly compute the function. Notice that the amplification method does not give an explicit construction of the circuits, since it uses a probabilistic argument.

Using the amplification method, Adleman [A] showed that every language in the probabilistic complexity class RP has polynomial-size Boolean circuits. Bennett and Gill [BG], also using the amplification method, extended Adleman's result by showing that every language in the complexity class BPP has polynomial-size circuits. Ajtai and Ben-Or [AB] used the amplification method to show that probabilistic constant-depth circuits can be simulated by deterministic constant-depth circuits, with only a polynomial increase in size.

In an elegant paper, Valiant [V] applied the amplification method to prove the existence of monotone Boolean formulas of size $O(n^{5.3})$ computing the majority function of n variables. Valiant's result is still the best upper bound known for monotone formulas computing majority. The question we study in this paper is whether or not Valiant achieved the best possible amplification. Our main results show that Valiant's amplification is indeed best possible, thus demonstrating a limit to the power of the amplification method. To state our results precisely, we need a few definitions.

DEFINITIONS. Given a Boolean function f from $\{0,1\}^n$ to $\{0,1\}$, define its *amplification function* A_f from $[0,1]$ to $[0,1]$ as follows: $A_f(p) = \Pr[f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = 1]$, where $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ are independent random variables that equal 1 with probability p and equal 0 with probability $1 - p$. A monotone Boolean function f *amplifies* (p,q) *to* (p',q') if $A_f(p) \leqslant p'$ and $A_f(q) \geqslant q'$.

The nice feature of amplifiers is that when composed with probabilistic formulas that approximately compute a function, they can provide much better approximations. We make this observation more precise in the simple lemma below.

DEFINITIONS. *A probabilistic Boolean formula* is a random variable with values ranging over Boolean formulas. A probabilistic Boolean formula $\mathbf{F}$ is a (p, q) *approximation* of a Boolean function g of n variables if for every string x in $\{0,1\}^n$,

(i) $g(x) = 0$ implies $\Pr[\mathbf{F}(x) = 1] \leqslant p$, and
(ii) $g(x) = 1$ implies $\Pr[\mathbf{F}(x) = 1] \geqslant q$.

LEMMA 1.1. *Suppose that F is a monotone Boolean formula of s variables that amplifies (p, q) to (p', q'), and that the independent probabilistic formulas $\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_s$ are (p, q) approximations of a Boolean function h. Then the composed formula $F \circ (\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_s)$ is a (p', q') approximation of h.*

Using this connection between amplifiers and probabilistic approximations, Valiant [V] derived his $O(n^{5.3})$ upper bound for majority. Valiant implicitly constructed the following two amplifiers.

DEFINITION. A *read-once formula* is a monotone formula in which every variable appears at most once.

THEOREM 1.2.

(a) *For every $m \geqslant 1$ and fixed $0 < p < 1$, there is a read-once formula of size $O(m^\alpha)$ that amplifies $(p, p + 1/m)$ to $(1/4, 3/4)$, where $\alpha = \log(2)/\log(\sqrt{5} - 1) \doteq 3.27$.*
(b) *For every $m \geqslant 2$, there is a read-once formula of size $O(m^2)$ that amplifies $(1/4, 3/4)$ to $(2^{-m}, 1 - 2^{-m})$.*

Our first theorem is a lower bound that shows that part (a) of Valient's construction is best possible up to a constant factor.

THEOREM 1.3. *If $0 < p < 1$ is fixed and $m \geqslant 1$, then every read-once formula that amplifies $(p, p + 1/m)$ to $(1/4, 3/4)$ must have size $\Omega(m^\alpha)$, where $\alpha = \log(2)/\log(\sqrt{5} - 1) \doteq 3.27$.*

Our next result is an upper bound that generalizes part (b) of Valiant's construction, by constructing good amplifiers from $(1/4, 3/4)$ to $(2^{-m_1}, 1 - 2^{-m_2})$. We also prove a lower bound showing that our construction and part (b) of Valiant's construction are best possible up to a constant factor.

THEOREM 1.4. Suppose $m_1, m_2 \geq 2$.

(a) *There is a read-once formula that amplifies* $(1/4, 3/4)$ *to* $(2^{-m_1}, 1 - 2^{-m_2})$ *having size* $O(m_1 m_2 + m_1 \log m_1 + m_2 \log m_2)$.

(b) *Every monotone formula that amplifies* $(1/4, 3/4)$ *to* $(2^{-m_1}, 1 - 2^{-m_2})$ *must have size* $\Omega(m_1 m_2 + m_1 \log m_1 + m_2 \log m_2)$.

An even stronger result to hope for would be an $\Omega(n^{\alpha+2})$ lower bound on the size of monotone formulas computing the majority function. Unfortunately, we have not been able to prove such a result. Khrapchenko [K] proved that formulas over the basis {AND, OR, NOT} require $\Omega(n^2)$ size to compute majority; no better lower bound for majority is known even for monotone formulas. Our paper shows instead lower bounds on a particular method for constructing monotone formulas.

As an application of the amplification method, we also show the existence of improved monotone Boolean formulas for threshold functions. Let $TH_{k,n}$ (the kth threshold function of n variables) be the Boolean function that is 1 if and only if at least k of its n variables are 1. We prove the existence of monotone formulas computing $TH_{k,n}$ of size $O(k^{4.3} n \log n)$. The best previous upper bound was $O(k^{8.3} n \log n)$, due to Friedman [F2], improved from [F1].

Related to the present paper is the paper by Moore and Shannon [MS]. Their model of computation is relay contact networks, which are more general than Boolean formulas. Moore and Shannon prove that contact networks can be designed to be arbitrarily reliable, no matter how unreliable the original components are. Along the way, they obtain an $\Omega(m_1 m_2)$ lower bound on the size of contact networks that amplify $(1/4, 3/4)$ to $(2^{-m_1}, 1 - 2^{-m_2})$. We can strengthen their lower bound to $\Omega(m_1 m_2 + m_1 \log m_1 + m_2 \log m_2)$, which is optimal. For more details, see Section 3 of the present paper.

The remainder of this paper is divided into four sections. In Section 2, we prove our lower bound on the amplification of $(p, p + 1/m)$ to $(1/4, 3/4)$. In Section 3, we present our results on the amplification of $(1/4, 3/4)$ to $(2^{-m_1}, 1 - 2^{-m_2})$. In Section 4, we give our $O(k^{4.3} n \log n)$ upper bound for threshold functions. Finally, in Section 5 we conclude with open problems.

2. AMPLIFYING $(p, p + 1/m)$ TO $(1/4, 3/4)$

The main goal of this section is to prove an $\Omega(m^{\alpha})$ lower bound on the size of read-once formulas that amplify $(p, p + 1/m)$ to $(1/4, 3/4)$, thus showing that part (a) of Valiant's construction is optimal. We also prove an $\Omega(m^2)$ lower bound on the size of monotone (not necessarily read-once) formulas that amplify $(p, p + 1/m)$ to $(1/4, 3/4)$.

The main idea of the proofs is to first show an upper bound on the derivative of the amplification function; this step is the hardest. Once we have an upper bound on the derivative, it will follow that amplification of $(p, p + 1/m)$ to $(1/4, 3/4)$ can be achieved only by large formulas. First we recall some standard definitions.

DEFINITIONS. A *monotone formula* of n variables is defined recursively as follows:

(i) for $1 \leqslant i \leqslant n$, the variables x_i are monotone formulas;
(ii) if f and g are monotone formulas, then $(f \wedge g)$ and $(f \vee g)$ are also monotone formulas.

A monotone formula *computes* a monotone Boolean function in the natural manner. The *size* of a monotone formula is the total number of occurrences of variables.

The theorem below gives an upper bound on the derivative of the amplification function. The proof relies on some independence properties of read-once formulas. For example, if two formulas f and g have distinct variables, then it is easy to see that $A_{f \wedge g} = A_f A_g$ and that $A_{f \vee g} = 1 - (1 - A_f)(1 - A_g)$.

THEOREM 2.1. *If f is a read-once formula and $0 < p < 1$, then*

$$A_f'(p) \leqslant [size(f)]^{1/\alpha} \frac{H[A_f(p)]}{H(p)},$$

where H is the entropy function $H(p) = -p \log_2(p) - (1 - p) \log_2(1 - p)$ and $\alpha = \log(2)/\log(\sqrt{5} - 1) \doteq 3.27$.

The proof of this theorem uses two inequalities that we state below. The first one, called Hölder's inequality, is well-known; a proof may be found in Hardy, Littlewood, and Pólya ([HLP], p. 24).

THEOREM 2.2. *Suppose that* $\alpha, \beta > 1$ *satisfy* $1/\alpha + 1/\beta = 1$, *and* $x_i, y_i \geqslant 0$ *for* $1 \leqslant i \leqslant n$. *Then*

$$\sum_{i=1}^{n} x_i y_i \leqslant \left(\sum_{i=1}^{n} x_i^{\alpha} \right)^{1/\alpha} \left(\sum_{i=1}^{n} y_i^{\beta} \right)^{1/\beta}.$$

The second inequality, dealing with the entropy function, is apparently new and is proved below.

THEOREM 2.3. *If* $0 < x, y \leqslant 1$, *then*

$$\left[\frac{H(x)}{x} \right]^{\beta} + \left[\frac{H(y)}{y} \right]^{\beta} \leqslant \left[\frac{H(xy)}{xy} \right]^{\beta},$$

where $\beta = \log(2)/\log((\sqrt{5}+1)/2) \doteq 1.44$.

Proof of Theorem 2.3. Let $f(x, y) = [H(xy)/xy]^{\beta} - [H(x)/x]^{\beta} - [H(y)/y]^{\beta}$. We will show that f is nonnegative on the region $(0, 1] \times (0, 1]$. The proof is divided into four cases, depending on the values of x and y.

Case 1: $x \geqslant 0.95$ *or* $y \geqslant 0.95$.

When $x = 1$ or $y = 1$, it is clear that $f(x, y) = 0$. Even when x or y is very close to 1, it is easy to see that $f(x, y) \geqslant 0$. In particular, by estimating f when x or y is close to 1, we can show that $x \geqslant 0.95$ or $y \geqslant 0.95$ implies that $f(x, y) \geqslant 0$.

Case 2: $x \leqslant 0.05$ *or* $y \leqslant 0.05$.

As x or y approaches 0, it is easy to see that $f(x, y)$ approaches ∞. In particular, by estimating f when x or y is close to 0, we can show that $x \leqslant 0.05$ or $y \leqslant 0.05$ implies that $f(x, y) \geqslant 0$.

Case 3: $0.55 \leqslant x \leqslant 0.65$ *and* $0.55 \leqslant y \leqslant 0.65$.

Let $a = [(\sqrt{5}-1)/2, (\sqrt{5}-1)/2] \doteq (0.62, 0.62)$. Observe that $f(a) = 0$ and that $(\partial f/\partial x)(a) = (\partial f/\partial y)(a) = 0$; thus a is a critical

point of f. The Hessian matrix of f at a is

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x^2}(a) & \frac{\partial^2 f}{\partial x \partial y}(a) \\ \frac{\partial^2 f}{\partial x \partial y}(a) & \frac{\partial^2 f}{\partial y^2}(a) \end{bmatrix} \doteq \begin{bmatrix} 0.804 & 0.353 \\ 0.353 & 0.804 \end{bmatrix}.$$

This matrix is positive definite, which implies that f has a local minimum at a. In fact, by estimating the third derivatives of f near a, we can show that the Hessian matrix of f is positive definite on the entire region $[0.55, 0.65] \times [0.55, 0.65]$. Thus f is convex and nonnegative on this region.

Case 4: All other x and y.

The remaining case will be proved with the assistance of a computer. Let G be the set of points (x, y) such that x and y are in $\{0.05, 0.06, \ldots, 0.95\}$, excluding those points in the region $[0.55, 0.65] \times [0.55, 0.65]$. Thus G has cardinality $(91)^2 - (11)^2 = 8160$. Using a computer, evaluate the function f on all the points of G. We find that f is nonnegative on G; in fact, it is bounded away from 0 by a nonnegligible amount. We next show, by a continuity argument, that f is nonegative for all x and y in Case 4. Suppose not; then there must be a critical point b such that $f(b) < 0$ and $(\partial f/\partial x)(b) = (\partial f/\partial y)(b) = 0$. Now some point c in G is very close to b. Estimating the second derivatives of f, we find that $f(c)$ is very close to $f(b) < 0$. But this contradicts our computation above showing that f is positive and bounded away from 0 on G. Thus Case 4 is established, and the proof is complete. □

With these two inequalities, we are now ready to prove Theorem 2.1 on the derivative of the amplification function.

Proof of Theorem 2.1. The proof is by induction on the size of the formula f. If $f = x_i$, then $A_f(p) = p$ and the inequality is clear. Suppose the desired inequality holds for two formulas g and h with distinct variables. Let $f = g \wedge h$, so that $A_f = A_g A_h$.

By differentiating and then using the induction hypothesis, we have

$$\begin{aligned} A_f'(p) &= A_f(p)\left[\frac{A_g'(p)}{A_g(p)} + \frac{A_h'(p)}{A_h(p)}\right] \\ &\leqslant \frac{A_f(p)}{H(p)}\left\{[size(g)]^{1/\alpha}\frac{H[A_g(p)]}{A_g(p)} + [size(h)]^{1/\alpha}\frac{H[A_h(p)]}{A_h(p)}\right\}. \end{aligned}$$

Applying Hölder's inequality (Theorem 2.2) to the right-hand side gives

$$\begin{aligned} A_f'(p) \leqslant \frac{A_f(p)}{H(p)}&[size(g) + size(h)]^{1/\alpha} \\ &\times \left\{\left[\frac{H[A_g(p)]}{A_g(p)}\right]^{\beta} + \left[\frac{H[A_h(p)]}{A_h(p)}\right]^{\beta}\right\}^{1/\beta}. \end{aligned}$$

Applying Theorem 2.3 and the identity $A_f = A_g A_h$ will show that

$$\begin{aligned} A_f'(p) &\leqslant \frac{A_f(p)}{H(p)}[size(f)]^{1/\alpha}\frac{H[A_g(p)A_h(p)]}{A_g(p)A_h(p)} \\ &= [size(f)]^{1/\alpha}\frac{H[A_f(p)]}{H(p)}. \end{aligned}$$

This establishes the inequality for $f = g \wedge h$. Similarly, we can prove the inequality for $f = g \vee h$, by using the identities $A_f = 1 - (1 - A_g)(1 - A_h)$ and $H(1 - x) = H(x)$. The induction is thus complete. □

Using the above bound on the derivative of the amplification function, we now prove our main result on amplifying $(p, p + 1/m)$ to $(1/4, 3/4)$.

COROLLARY 2.4. *If $0 < p < 1$ is fixed and $m \geqslant 1$, then every read-once formula that amplifies $(p, p + 1/m)$ to $(1/4, 3/4)$ must have size $\Omega(m^{\alpha})$, where $\alpha = \log(2)/\log(\sqrt{5} - 1) \doteq 3.27$.*

Proof. Suppose f is a read-once formula amplifying $(p, p + 1/m)$ to $(1/4, 3/4)$. By the mean value theorem,

$$\tfrac{1}{2} \leqslant A_f\left(p + \frac{1}{m}\right) - A_f(p) = \frac{1}{m}A_f'(\eta),$$

for some η satisfying $p < \eta < p + 1/m$. But by Theorem 2.1 we have

$$A_f'(\eta) \leqslant \frac{[size(f)]^{1/\alpha}}{H(\eta)}.$$

Combining the previous two inequalities shows that

$$size(f) \geqslant \left[\frac{mH(\eta)}{2}\right]^{\alpha} = \Omega(m^{\alpha}).$$

This establishes the result. □

We can prove a similar result for monotone formulas that are not necessarily read-once, but the lower bound now is only $\Omega(m^2)$. The proof depends on the following theorem, which shows that the maximum of $A_f'(p)$, over all Boolean functions f of n variables, occurs when f is a threshold function.

THEOREM 2.5. *If f is a Boolean function of n variables, then*

$$|A_f'(p)| \leqslant n\binom{n-1}{t-1} p^{t-1}(1-p)^{n-t},$$

where $t = \lceil np \rceil$.

Proof. By the definition of A_f, we have

$$A_f(p) = \sum_{k=0}^{n} a_k p^k (1-p)^{n-k},$$

where a_k is the number of binary strings x with exactly k 1's satisfying $f(x) = 1$. Notice that $0 \leqslant a_k \leqslant \binom{n}{k}$. After differentiating, we obtain

$$A_f'(p) = \sum_{k=0}^{n} a_k p^{k-1}(1-p)^{n-k-1}(k-np).$$

Thus to maximize $A_f'(p)$, we should choose $a_k = 0$ for $k < np$ and $a_k = \binom{n}{k}$ for $k > np$. For this choice of a_k we can use the following

identity (which can be proved by backward induction on t):

$$\sum_{k=t}^{n} \binom{n}{k} p^{k-1}(1-p)^{n-k-1}(k-np) = n\binom{n-1}{t-1} p^{t-1}(1-p)^{n-t}.$$

Here we choose $t = \lceil np \rceil$. Similarly, to minimize $A_f'(p)$, we obtain the negation of the above expression, which proves the theorem. □

Notice that if $0 < p < 1$ is fixed and n approaches infinity, the right-hand side of the above inequality is asymptotic to $\sqrt{n/[2\pi p(1-p)]}$, by Stirling's formula. We obtain the following corollary.

COROLLARY 2.6. *If $0 < p < 1$ is fixed and $m \geq 1$, then every monotone formula that amplifies $(p, p + 1/m)$ to $(1/4, 3/4)$ must have size $\Omega(m^2)$.*

Proof. Suppose f is a monotone formula that amplifies $(p, p + 1/m)$ to $(1/4, 3/4)$. By the mean value theorem, we have $A_f'(\eta) \geq m/2$ for some η satisfying $p < \eta < p + 1/m$. But Theorem 2.5 shows that $A_f'(\eta)$ is $O(\sqrt{n})$, where n is the number of variables on which f depends. Since the size of f is at least n, we have $size(f) = \Omega(m^2)$. □

3. AMPLIFYING $(1/4, 3/4)$ TO $(2^{-m_1}, 1 - 2^{-m_2})$

In this section we give matching upper and lower bounds on the size of read-once formulas that amplify $(1/4, 3/4)$ to $(2^{-m_1}, 1 - 2^{-m_2})$. Recall that Valiant [V] had constructed read-once formulas of size $O(m^2)$ that amplify $(1/4, 3/4)$ to $(2^{-m}, 1 - 2^{-m})$. We will extend Valiant's construction by amplifying $(1/4, 3/4)$ to $(2^{-m_1}, 1 - 2^{-m_2})$. We will then prove a lower bound showing the optimality of the construction.

To begin with, we present our extension of Valiant's construction. The extension is used in Section 4 to obtain small monotone formulas for threshold functions.

THEOREM 3.1. *For $m_1, m_2 \geq 2$, there is a read-once formula that amplifies $(1/4, 3/4)$ to $(2^{-m_1}, 1 - 2^{-m_2})$ having size $O(m_1 m_2 + m_1 \log m_1 + m_2 \log m_2)$.*

Proof. Assume, without loss of generality, that $m_1 \geqslant m_2$. Choose j and k to satisfy $jk = m_1$ and $(1 - 2^{-j})^k = 1 - 2^{-m_2}$. Solving these equations, we find that $j = \Theta(m_2 + \log m_1)$ and $k = \Theta[m_1/(m_2 + \log m_1)]$. By Valiant's construction (Theorem 1.2), there is a read-once formula of size $O(j^2)$ that amplifies $(1/4, 3/4)$ to $(2^{-j}, 1 - 2^{-j})$. Taking the AND of k disjoint copies of such amplifiers, we obtain a read-once formula that amplifies $(1/4, 3/4)$ to $[2^{-jk}, (1 - 2^{-j})^k] = (2^{-m_1}, 1 - 2^{-m_2})$. The total size is $O(kj^2) = O[m_1(m_2 + \log m_1)]$, so we are done. □

The term that dominates the above upper bound depends on whether or not m_1 and m_2 are within an exponential of each other.

We will now prove that the above construction is optimal. In fact, we will show the construction is optimal over a wider class of computational devices, called relay contact networks. We need some definitions.

DEFINITIONS. A *monotone contact network* is a directed graph $G = (V, E)$ that has two distinguished vertices s and t, and has labels on its edges. The edge labels are chosen from the set of variables $\{x_1, x_2, \ldots, x_n\}$. A monotone contact network N *computes* a Boolean function f_N from $\{0,1\}^n$ to $\{0,1\}$ as follows. Given a string y in $\{0,1\}^n$, each edge of N is set to 0 or 1 according to the value of its label. Then $f_N(y) = 1$ if there is a path from s to t using only edges with value 1, and $f_N(y) = 0$ otherwise. We will often identify a contact network with the function that it computes. The *size* of a contact network is the number of edges in it. A *read-once* contact network is a monotone contact network in which every variable appears in at most one label.

Notice that a Boolean formula is essentially a series-parallel contact network, with ANDs represented in series, and ORs represented in parallel. We recall two theorems due to Moore and Shannon [MS].

DEFINITIONS. Given a monotone Boolean function f, its *length* $l(f)$ is the fewest number of 1's in a string x such that $f(x) = 1$. Its *width* $w(f)$ is the fewest number of 0's in a string x such that $f(x) = 0$.

THEOREM 3.2. *If f is a monotone Boolean function, then*

$$p^{l(f)} \leqslant A_f(p) \leqslant 1 - (1-p)^{w(f)}.$$

THEOREM 3.3. *If N is a monotone contact network, then* $size(N) \geqslant l(N)w(N)$.

To prove our lower bound on the size of contact network amplifiers, we need the following lemma, which is interesting on its own.

DEFINITIONS. A *k-conjunctive normal form* (k-CNF) *formula* is an AND of ORs of literals, where each OR has fanin at most k. A *k-disjunctive normal form* (k-DNF) *formula* is an OR of ANDs of literals, where each AND has fanin at most k.

LEMMA 3.4.

(*a*) *If f is a monotone k-CNF formula, then*

$$A_f(p) \leqslant [1 - (1-p)^k]^{l(f)}.$$

(*b*) *If f is a monotone k-DNF formula, then*

$$A_f(p) \geqslant 1 - [1 - p^k]^{w(f)}.$$

Proof. We will prove only part (a), as part (b) will follow by duality. The proof is by induction on the number of clauses in f. If $f \equiv 1$, then the inequality is clear. Otherwise, suppose by induction that the inequality holds for all monotone k-CNF formulas with fewer clauses than f has. Assume, without loss of generality, that $f = (x_1 \vee x_2 \vee \cdots \vee x_j)g$, where $j \leqslant k$ and g is a monotone k-CNF formula. Let g_i be the formula obtained when, in formula g, the variables $x_1, x_2, \ldots, x_{i-1}$ are set to 0 and the variable x_i is set to 1. It is easy to see that

$$A_f(p) = pA_{g_1}(p) + p(1-p)A_{g_2}(p) + \cdots + p(1-p)^{j-1}A_{g_j}(p).$$

By the induction hypothesis, we have

$$A_{g_i}(p) \leqslant [1 - (1-p)^k]^{l(g_i)} \leqslant [1 - (1-p)^k]^{l(f)-1}.$$

Substituting this inequality into the previous one yields

$$\begin{aligned} A_f(p) &\leqslant [p + p(1-p) + \cdots + p(1-p)^{j-1}][1-(1-p)^k]^{l(f)-1} \\ &= [1-(1-p)^j][1-(1-p)^k]^{l(f)-1} \\ &\leqslant [1-(1-p)^k]^{l(f)}. \end{aligned}$$

The induction is thus complete. □

THEOREM 3.5. *For $m_1, m_2 \geqslant 2$, every monotone contact network that amplifies $(1/4, 3/4)$ to $(2^{-m_1}, 1-2^{-m_2})$ must have size $\Omega(m_1 m_2 + m_1 \log m_1 + m_2 \log m_2)$.*

Proof. Suppose N is a monotone contact network that amplifies $(1/4, 3/4)$ to $(2^{-m_1}, 1-2^{-m_2})$. Theorem 3.2 implies that $l(N) \geqslant m_1/2$ and that $w(N) \geqslant m_2/2$. Thus, by Theorem 3.3, the size of N is $\Omega(m_1 m_2)$.

We next show that the size of N is $\Omega(m_2 \log m_2)$. Let k be a parameter to be chosen later. Consider the monotone Boolean function f of n variables that is 1 iff there is a path P from s to t in network N such that all edges of P have been set to 1, and the number of different labels on P is at most k. Clearly f is expressible as a monotone k-DNF formula. By Lemma 3.4, we have $A_f(p) \geqslant 1-[1-p^k]^{w(f)}$. Since $A_f(1/4) \leqslant A_N(1/4) \leqslant 1/4$, the previous inequality implies that $w(f) \leqslant 4^k \log(4/3) \leqslant 4^k$. Thus by setting at most 4^k variables to 0, all paths from s to t using at most k variables can be eliminated.

Now comes the main idea of the proof. Let N' be the network formed when the above (at most 4^k) variables are set to 0. Then $l(N') \geqslant k+1$ and $w(N') \geqslant w(N) - 4^k$. Thus, by Theorem 3.3, the size of N' is at least $(k+1)[w(N) - 4^k]$. Choose $k = \lfloor \log[w(N)/2] / \log(4) \rfloor$. With this choice of k, the size of N' is seen to be $\Omega[w(N) \log w(N)]$. Since $w(N) \geqslant m_2/2$, the size of N' (and hence the size of N) is $\Omega(m_2 \log m_2)$.

By a dual argument, the size of N can be shown to be $\Omega(m_1 \log m_1)$. This will complete the proof. □

The above result generalizes a result in a preliminary version of this paper ([B], Theorem 3.5), which applied only to read-once formulas.

4. THRESHOLD FUNCTIONS

In this section we use the amplification method to prove the existence of small monotone formulas for threshold functions. Recall that the function $TH_{k,n}$ (the kth threshold function of n Boolean variables) equals 1 if and only if at least k of its n variables are 1. Friedman ([F2], improved from [F1]) proved the existence of monotone formulas computing $TH_{k,n}$ of size $O(k^{8.3} n \log n)$. We will present an improved bound of $O(k^{4.3} n \log n)$. Neither result gives explicit formulas, since both use probabilistic arguments.

Our approach is as follows: we first prove, in Theorem 4.2, the existence of small approximate formulas for $TH_{k,n}$. In Corollary 4.3, we show how to use such approximate formulas to obtain small exact formulas for $TH_{k,n}$.

Recall the definitions given in Section 1 of probabilistic formulas and (p, q) approximations. We need two additional definitions.

DEFINITIONS. Given a monotone Boolean function f, let $min(f)$ denote those strings x that minimally satisfy $f(x) = 1$ [i.e., $f(x) = 1$ but no $x' < x$ satisfies $f(x') = 1$]. Let $max(f)$ denote those strings x that maximally satisfy $f(x) = 0$.

As an example of these definitions, observe that $min(TH_{k,n})$ is the kth level of the n-dimensional Boolean cube, and that $max(TH_{k,n})$ is the $(k-1)$th level. In particular, $min(TH_{k,n})$ has cardinality $\binom{n}{k}$, and $max(TH_{k,n})$ has cardinality $\binom{n}{k-1}$.

LEMMA 4.1. *Suppose there is a monotone probabilistic formula of size s that is a (p, q) approximation of a Boolean function f.*

(a) *If $p|max(f)| + (1-q)|min(f)| < 1$, then there is a monotone formula of size s that computes f.*

(b) *If $(1-q)|min(f)| < 1$, then there is a monotone probabilistic formula of size s that is a $(p', 1)$ approximation of f, where*

$$p' = \frac{p}{1-(1-q)|min(f)|}.$$

(c) *If $p|max(f)| < 1$, then there is a monotone probabilistic formula of size s that is a $(0, q')$ approximation of f, where*

$$q' = 1 - \frac{1-q}{1-p|max(f)|}.$$

Proof. Let **F** be the monotone probabilistic formula promised in the hypothesis.

(a) The probability that **F** does not compute f is

$$\begin{aligned}\Pr[\mathbf{F} \text{ doesn't compute } f] &\leqslant \sum_{x \in max(f)} \Pr[\mathbf{F}(x) = 1] \\ &\quad + \sum_{x \in min(f)} \Pr[\mathbf{F}(x) = 0] \\ &\leqslant |max(f)|p + |min(f)|(1-q) \\ &< 1.\end{aligned}$$

Thus, **F** computes f with positive probability, which settles part (a).

(b) We first show that $\mathbf{F} \geqslant f$ with decent probability. The probability that $\mathbf{F} \geqslant f$ does not hold is

$$\begin{aligned}\Pr[\neg(\mathbf{F} \geqslant f)] &\leqslant \sum_{x \in min(f)} \Pr[\mathbf{F}(x) = 0] \\ &\leqslant |min(f)|(1-q).\end{aligned}$$

Thus, $\mathbf{F} \geqslant f$ holds with probability at least $1 - (1-q)|min(f)|$. Define the probabilistic formula **G** to be **F** restricted to those formulas for which $\mathbf{F} \geqslant f$. If a string x satisfies $f(x) = 1$, then

$$\Pr[\mathbf{G}(x) = 1] = \Pr[\mathbf{F}(x) = 1 : \mathbf{F} \geqslant f] = 1.$$

On the other hand, if $f(x) = 0$, then

$$\begin{aligned}\Pr[\mathbf{G}(x) = 1] &= \Pr[\mathbf{F}(x) = 1 : \mathbf{F} \geqslant f] \\ &\leqslant \frac{\Pr[\mathbf{F}(x) = 1]}{\Pr[\mathbf{F} \geqslant f]} \\ &\leqslant \frac{p}{1-(1-q)|min(f)|} \\ &= p'.\end{aligned}$$

Thus, **G** is a $(p', 1)$ approximation of f, which settles part (b).

(c) The proof of part (c) is dual to that of part (b). □

THEOREM 4.2. *For every k and n, there is a monotone probabilistic formula of size $O[k^{\alpha}(\log k)^2 n]$ that is a $(0, 1 - 1/k)$ approximation of $TH_{k,n}$.*

Proof. We first prove the result for $k \geqslant n^{1/3}$. Let **F** be the probabilistic formula that equals x_i with probability $1/n$, for all $1 \leqslant i \leqslant n$. It is easy to see that **F** is a $[(k-1)/n, k/n]$ approximation of $TH_{k,n}$. Taking the OR of n/k such formulas, we obtain a $(1 - 1/e, 1 - 1/e + 1/k)$ approximation of $TH_{k,n}$. By Valiant's construction (Theorem 1.2), we obtain a $(1/4, 3/4)$ approximation of $TH_{k,n}$ with a factor of $O(k^{\alpha})$ more work. By applying Theorem 3.1, we obtain a $[1/(10n^k), 1 - 1/(10k)]$ approximation of $TH_{k,n}$ with a factor of $O(k \log k \log n)$ more work. Lemma 4.1 then provides us with a $(0, 1 - 1/k)$ approximation of $TH_{k,n}$. The total size is

$$O\left[\frac{n}{k} k^{\alpha}(k \log k \log n)\right] = O[(k^{\alpha} \log k) n \log n] = O[k^{\alpha}(\log k)^2 n],$$

which establishes the theorem for $k \geqslant n^{1/3}$.

We next prove the result for $k \leqslant n^{1/3}$. Let $l = k^3$. By the above paragraph, there is a monotone probabilistic formula **G** of size $O[k^{\alpha}(\log k)^2 l]$ that is a $(0, 1 - 1/k)$ approximation of $TH_{k,l}$. Suppose the variables of **G** are $y_1, y_2, \ldots, y_l$. Let $x_1, x_2, \ldots, x_n$ be n other variables. Randomly partition the x_i into l blocks of n/l variables each. Substitute for each y_i in **G** the OR of the n/l variables in the ith block. Call the resulting probabilistic formula **H**. Then **H** is a $(0, q)$ approximation of $TH_{k,n}$, where

$$\begin{aligned} q &= \left(1 - \frac{1}{k}\right)\left(\frac{n}{l}\right)^k \frac{\binom{l}{k}}{\binom{n}{k}} \\ &\geqslant \left(1 - \frac{1}{k}\right)\left(1 - \frac{1}{l}\right)\left(1 - \frac{2}{l}\right) \cdots \left(1 - \frac{k-1}{l}\right) \\ &\geqslant \left(1 - \frac{1}{k}\right)\left(1 - \frac{k^2}{l}\right) \\ &\geqslant 1 - \frac{2}{k}. \end{aligned}$$

The formula **H** has size

$$O\left\{[k^{\alpha}(\log k)^2 l] \frac{n}{l}\right\} = O[k^{\alpha}(\log k)^2 n].$$

With a constant factor more work, we can make **H** a $(0, 1 - 1/k)$ approximation of $TH_{k,n}$, which completes the proof. □

Now we can show the $O(k^{4.3} n \log n)$ bound for exactly computing $TH_{k,n}$.

COROLLARY 4.3. *For every k and n, there exists a monotone formula of size*

$$O[(k^{\alpha+1} \log k) n \log n] = O(k^{4.3} n \log n)$$

that computes $TH_{k,n}$.

Proof. Theorem 4.2 shows the existence of a monotone probabilistic formula of size $O[k^{\alpha}(\log k)^2 n]$ that is a $(0, 1 - 1/k)$ approximation of $TH_{k,n}$. Taking the OR of $10k \log n / \log k$ independent copies of such formulas, we obtain a $(0, 1 - n^{-10k})$ approximation of $TH_{k,n}$. Appealing to Lemma 4.1, we obtain the existence of monotone formulas computing $TH_{k,n}$ exactly that have size

$$O\left\{[k^{\alpha}(\log k)^2 n] \frac{k \log n}{\log k}\right\} = O[(k^{\alpha+1} \log k) n \log n].$$

This completes the proof. □

Thus we see that the amplification method yields small monotone formulas for threshold functions.

5. CONCLUSIONS

We have proved limits on the power of the amplification method for probabilistic Boolean formulas, showing that Valiant's method of amplification [V] is optimal. In particular, we showed that every read-once formula that amplifies $(p, p + 1/m)$ to $(1/4, 3/4)$ requires size $\Omega(m^{\alpha})$, where $\alpha = \log(2)/\log(\sqrt{5} - 1) \doteq 3.27$. Every monotone contact network that amplifies $(1/4, 3/4)$ to $(2^{-m_1}, 1 - 2^{-m_2})$ requires size $\Omega(m_1 m_2 + m_1 \log m_1 + m_2 \log m_2)$. Using the amplification method, we also provided monotone formulas of size $O(k^{4.3} n \log n)$ for the kth threshold function of n variables.

There are several open problems remaining:

1. Is Corollary 2.4 on amplifying $(p, p + 1/m)$ to $(1/4, 3/4)$ true for arbitrary monotone formula amplifiers? If not, then there would exist monotone formulas for majority smaller than the best upper bound known of $O(n^{5.3})$.
2. Can either Valiant's $O(n^{5.3})$ majority bound or our $O(k^{4.3} n \log n)$ bound for $TH_{k,n}$ be achieved by explicit formulas? Both bounds were proved using probabilistic methods. The only explicit monotone formulas of polynomial size known for majority use the $O(\log n)$ depth sorting network of Ajtai, Komlós, and Szemerédi [AKS], but the degree of the polynomial is huge. Friedman [F1] showed that monotone formulas for $TH_{k,n}$ of size $O[poly(k) n \log n]$ can be explicitly constructed using the above sorting network, but again the polynomial in k has huge degree.
3. Can lower bounds better than $\Omega(n^2)$ be obtained for the size of monotone formulas computing majority? Perhaps the methods of this paper provide a step toward this goal.

ACKNOWLEDGMENTS

I would like to thank my adviser Michael Sipser for valuable discussions. I thank Nicholas Pippenger for informing me of the paper by Moore and Shannon [MS]. This research was supported in part by a National Science Foundation Graduate Fellowship, and by Grant AFOSR-82-0326. A preliminary version of this paper appeared in [B].

NOTES

1. Current address is Department of Computer Science, New York University, 251 Mercer Street, New York, NY 10012.

REFERENCES

[A] L. Adleman, "Two theorems on random polynomial time," *Proc. 19th IEEE Symp. Found. Computer Sci.* 75–83 (1978).

[AB] M. Ajtai and M. Ben-Or, "A theorem on probabilistic constant depth computations," *Proc. 16th ACM Symp. Theory Computing* 471–474 (1984).

[AKS] M. Ajtai, J. Komlós, and E. Szemerédi, "Sorting in $c \log n$ parallel steps," *Combinatorica* 3: 1–19 (1983). See also "An $O(n \log n)$ sorting network," *Proc. 15th ACM Symp. Theory Computing* 1–9 (1983).

[BG] C. G. Bennett and J. Gill, "Relative to a random oracle A, $P^A \neq NP^A \neq co\text{-}NP^A$ with probability 1," *SIAM J. Computing* 10: 96–113 (1981).

[B] R. B. Boppana, "Amplification of probabilistic Boolean formulas," *Proc. 26th IEEE Symp. Found. Computer Sci.* 20–29 (1985).

[F1] J. Friedman, "Constructing $O(n \log n)$ size monotone formulae for the kth threshold function of n Boolean variables," *SIAM J. Computing* 15:3, 641–654 (1986).

[F2] J. Friedman, personal communication, August 1984.

[HLP] G. H. Hardy, J. E. Littlewood, and G. Pólya, *Inequalities*, Cambridge University Press, Cambridge, 1952.

[K] V. M. Khrapchenko, "A method of determining lower bounds for the complexity of Π-schemes," *Mat. Zamet.* 10:1, 83–92 (1971). English translation in *Math. Notes Acad. Sci. USSR* 474–479 (1972).

[MS] E. F. Moore and C. E. Shannon, "Reliable circuits using less reliable relays," *J. Franklin Inst.* 262: 191–208 and 281–297 (1956).

[V] L. G. Valiant, "Short monotone formulae for the majority function," *J. Algorithms* 5: 363–366 (1984).

PARALLEL TREE CONTRACTION[1]

PART 1: FUNDAMENTALS

Gary L. Miller and John H. Reif

ABSTRACT

This paper introduces parallel tree contraction: a new bottom-up technique for constructing parallel algorithms on trees. Contraction can be used to solve a wide variety of problems. Two examples included in this article are expression evaluation and subexpression elimination. In this paper we show these applications only require $O(\log n)$ time and $O(n/\log n)$ processors on a 0-sided randomized PRAM or $O(n)$ processors on a deterministic PRAM. In the process of finding these efficient algorithms we find efficient parallel algorithms for several other problems including generating random permutation in parallel and randomized techniques for work load balancing on PRAMs.

We have found other application of parallel tree contraction including testing isomorphism of trees, canonical forms for trees,

Advances in Computing Research, Volume 5. pages 47–72.

ISBN: 0–89232–896–7

constructing planar embeddings, and testing isomorphism of planar graphs. These applications appear in a companion paper [MR2].

1. INTRODUCTION

1.1. Top-Down vs Bottom-Up Tree Algorithms

Trees play a fundamental role in many computations, both for sequential as well as parallel problems. The classic paradigm applied to generate parallel algorithms in the presence of trees has been divide-conquer; finding a vertex which is a "1/3–2/3" separator, and recursively solving the two subproblems. A now classic example is Brent's work on parallel evaluation of arithmetic expressions [B]. This "top-down" approach has several complications, one of which is finding the separators that must separate the tree into components with size $\leqslant 2/3$ of the original size. We define *dynamic expression evaluation* as the task of evaluating the expression with no free preprocessing. If we apply Brent's method, finding the separators seem to add a factor of $\log n$ to the running time.

We give a "bottom-up" algorithm to handle trees. That is, all modifications to the tree are done locally. This "bottom-up" approach, which we call *CONTRACT*, has two major advantages over the "top-down" approach: (1) The control structure is straightforward and easier to implement facilitating new algorithms using fewer processors and less time. (2) Problems for which it was too difficult or too complicated to find polylog parallel algorithms are now easy. We believe our lasting contribution will be CONTRACT. It has already been applied to finding small separators for planar graphs in parallel [M] as well as numerous applications appearing in the companion paper [MR2].

1.2. The PRAM Model

We shall use the *PRAM* model of a parallel processing device (see [SV]). A PRAM consists of a collection of processors. Each processor is a random access machine that can read and write in a common random access memory. In unit time these processors are allowed concurrent reads and concurrent writes (CRCW), as well as arithmetic operations on integers of magnitude upper bounded by $n^{O(1)}$. There are two natural implementations of concurrent writes. (1) If

two or more processors attempt to write in a given location of common memory then one of the processors will succeed. The performance of the algorithm should not depend on which processor succeeds. (2) In the second model concurrent writes in a given location cause detectable noise to be stored in that location. Unless otherwise stated we shall assume the first model for concurrent writes. But all of our algorithms also work with the same performance in the second model.

Many of our algorithms use randomization. That is, each processor has access to an independent random number of magnitude $\leqslant n$ per step. A (*1-sided*) *randomized algorithm* A is said to accept a language L in $T(n)$ time using $P(n)$ processors if the following conditions hold: (1) on all inputs w of length n, A uses at most $T(n)$ time and $P(n)$ processors independent of the random bits; (2) if A rejects w then $w \notin L$; (3) if A accepts w then with probability of error at most $1/n$ we can conclude that $w \in L$. Note that we have chosen $1/n$ for our error bound instead of the common value $1/2$. It generally seems to increase the running time by a factor of $\log n$ to achieve the error bound $1/n$ from an algorithm with error bound $1/2$. On the other hand, given an algorithm with error bounded by $1/n$ if we increase the running time by a constant factor of α we can achieve the tighter error bound $1/n^{\alpha}$. We say an algorithm is *0-sided randomized* if it is always correct when it terminates and the probability of termination is at least $1 - 1/n$. We often denote 0-sided and 1-sided by subscripts of 0 and 1 respectively (see [R1]).

All our PRAM algorithms will only use a polynomial number of processors. We shall take considerable effort to minimize the number of processors used.

1.3. Expression Evaluation and Our Results

Arithmetic expression evaluation is a good robust problem exhibiting our techniques. An arithmetic expression is a tree where the leaves have values from some domain and each internal vertex has two children and a label from $\{+, \times, \div\}$. We assume that these binary operations can be performed in constant time. As we shall see in the companion paper, these techniques are very general but most of the ideas will be well illustrated in the case of expression evaluation.

We exhibit a deterministic PRAM algorithm for dynamic expression evaluation using $O(\log n)$ time and $O(n)$ processors and a 0-sided randomized version of this algorithm using only $O(n/\log n)$ processors. We then extend these algorithms to evaluate all subexpressions using the same time and number of processors. In comparison Brent [B] showed that expressions of size n could be transformed into straight-line code of depth $O(\log n)$. Dynamic transformation of code in parallel by Brent's method seems to require $\Omega(\log^2 n)$ time.

We also give an algorithm that uses the same resource bounds as the expression case for computing the value of all subexpressions. This result is a natural generalization of parallel prefix evaluation [F, LF, V]. Up to constant factors we use no more time or processors. The list-ranking problem (see [V]) is also a special case. Thus, we have given an $O(\log n)$ time optimal algorithm for list-ranking. This is the first known $O(\log n)$ time and $n/\log n$ processor algorithm for the problem.

1.4. Organization of This Paper

The body of the paper consists of 6 sections. In Section 2 we define two abstract operations on trees, RAKE and COMPRESS. We show that only $O(\log n)$ simultaneous applications of these operations are needed to reduce a tree to a point. In Section 3 we give both deterministic as well as randomized implementations of RAKE and COMPRESS both of these implementations reduce a tree to a point in $O(\log n)$ time using $O(n)$ processors. In Section 4 we show how to implement these operations on a randomized PRAM such that any tree is reduced to a point in $O(\log n)$ time using an optimal number of processors. We call this implementation *dynamic tree contraction*. In Section 4 we also describe how to generate a random permutation in $O(\log n)$ time using only $n/\log n$ processors and how to remove a constant proportion of the zeros from a random string in $O[\log(\log n)]$ time using only $n/\log(\log n)$ processors. These operations give a general technique for minimizing the number of processors used. We call this general technique *processor work load balancing*. In Sections 5 we apply dynamic tree contraction to expression evaluation, subexpression evaluation and related tree problems. In Section 6 we show that a natural modification of parallel tree contraction, which we call *asynchronous parallel tree contraction*, works in $O(\log n)$ time even

if the cost of raking k siblings is $O(\log k)$ time. In Section 7 we give a partial analysis of the random variable *MATE* that arises in the randomized parallel tree contraction algorithms.

2. THE RAKE AND COMPRESS OPERATIONS

Let $T = (V, E)$ be a rooted tree with n vertices and root r. We describe two simple parallel operations on T such that at most $O(\log n)$ applications are needed to reduce T to a single vertex.

Let *RAKE* be the operation that removes all leaves from T. It is easy to see that RAKE may need to be applied a linear number of times to a highly unbalanced tree to reduce T to a single vertex. We can circumvent this problem by adding one more operation.

We say a sequence of vertices $v_1, \ldots, v_k$ is a *chain* if v_{i+1} is the only child of v_i for $1 \leqslant i \leqslant k$, and v_k has exactly one child and that child is not a leaf. The chain is said to have length k. In one parallel step, we *compress* a chain by identifying v_i with v_{i+1} for i odd and $1 \leqslant i < k$. Thus, the chain $v_1, \ldots, v_k$ is replaced with a chain $v'_1, \ldots, v'_{\lfloor k/2 \rfloor}$. Let *COMPRESS* be the operation on T that "compresses" all maximal chains of T in one step. Note that a maximal chain of length one is not affected by COMPRESS.

Let *CONTRACT* be the simultaneous application of RAKE and COMPRESS to the entire tree. We next show that the CONTRACT operation needs only be executed $O(\log n)$ times to reduce T to *its root*.

THEOREM 2.1. *After* $\lfloor \log_{5/4} n \rfloor$ *applications of CONTRACT to a tree on n vertices it is reduced to its root.*

Proof. We partition the vertices of T into two sets *Ra* and *Com* such that $|Ra|$ will decrease by a factor of 4/5 after an execution of RAKE and $|Com|$ will decrease by a factor of 1/2 after COMPRESS.

Let V_0 be the leaves of T, V_1 be the vertices with only one child, and let V_2 be those vertices with two or more children. We further partition the set V_1 into C_0, C_1, and C_2 according to whether the child is in V_0, V_1, and V_2, respectively. Similarly we partition the vertices C_1 into GC_0, GC_1, and GC_2 corresponding to whether the grandchild is in V_0, V_1, and V_2, respectively. Let $Ra = V_0 \cup V_2 \cup C_0 \cup C_2 \cup GC_0$ and $Com = V - Ra$.

To see that the size of *Ra* decreases by a factor of 1/5 after each RAKE we show that $|Ra| \leqslant 5|V_0|$. The inequality follows by observing the following inequalities: $|V_2| < |V_0|$, $|C_0| \leqslant |V_0|$, $|GC_0| \leqslant |V_0|$, and $|C_2| \leqslant V_2|$.

Note that all vertices in V_1 except those of C_0 belong to a chain. Thus, every vertex of *Com* belongs to some maximal chain. If $v_1, \ldots, v_k$ are the vertices of a maximal chain then either $v_k \in C_2$ or $v_k \in GC_0$. In either case $v_1, \ldots, v_{k-1}$ are the only elements in the chain belonging to *Com*. Thus, the number of elements in a maximal chain of *Com* decreases by at least a factor of 1/2 after COMPRESS. □

The type of argument used in the proof of Theorem 2.1 will be used in the analysis of several other algorithms which are based on CONTRACT. Given a tree $T = (V, E)$ let $Rake(V) = Ra$ and $Compress(V) = Com$ as defined in the above proof.

There are many useful applications of parallel tree contraction and expansion. For each given application, we associate a certain procedure with each RAKE and COMPRESS operation that we assume can be computed in parallel quickly. Typically the vertices of the tree T will contain labels storing information relevant to the given application. The RAKE and COMPRESS operations will modify these labels, as well as the tree itself. To apply parallel tree contraction to a problem seems to require finding a general form for implementing and storing the composition of unary functions.

As a simple example we consider the case when T is an expression tree over $\{+, \times\}$ the RAKE corresponds to the operation of (1) evaluating a vertex if all of its children have been evaluated or (2) partially evaluating a vertex if some of its children have been evaluated. The cost of applying RAKE to an expression tree is the cost of evaluating a vertex. If a vertex has been partially evaluated except for one child then the value of the vertex is a *linear function*, say, $aX + b$ where X is a variable of the remaining child, and a and b are scalars over some semiring. Thus a chain is a sequence of vertices each of which is a linear function of its child. In this application, COMPRESS is simply pairwise composition of linear functions. Thus, in this example the only nontrivial observation is the fact that linear functions in one argument are closed under composition and each linear function can be represented by two scalars.

This gives a simple proof that (after preprocessing) expressions can be evaluated in time $O(\log n)$ using $O(n)$ processors on a PRAM. On the other hand, the naive dynamic implementation of COMPRESS requires $O(\log n)$ time since we first will determine the parity of each vertex on a chain by pointer jumping, e.g. (doubling-up), then combine consecutively the odd and even vertices pairwise in constant time. In the next section we implement both a deterministic and a randomized variant of COMPRESS that can be performed in constant time.

3. DYNAMIC TREE CONTRACTION (DETERMINISTIC AND RANDOMIZED)

In this section we describe in more detail two implementations of COMPRESS. The first is deterministic while the second is a randomized algorithm (see Section 3.2). The deterministic algorithm seems to need $O(n)$ processors to achieve $O(\log n)$ time. We will show in Section 4 how to improve the randomized algorithm to use only $O(n/\log n)$ processors and $O(\log n)$ time. In this section we assume that the trees are of bounded degree. The analysis of parallel tree contraction on trees of unbounded degree is in Section 6.

3.1. Deterministic Tree Contraction

Let T be a rooted tree with vertex set V of size $n = |V|$ and root $r \in V$. We view each vertex, which is not a leaf, as a function to be computed where the children supply the arguments. For each vertex v with children $v_1 \ldots v_k$ we will set aside k locations $l_1 \ldots l_k$ in common memory. Initially each l_i is empty or *unmarked*. When the value of v_i is known we will assign it to l_i: this will be simply denoted by *mark* l_i. Let $Arg(v)$ denote the number of unmarked l_i. Thus, initially $Arg(v) = k$, the number of children of v. We need one further notation: Let $vertex[P(v)]$ be the vertex associated with the sole parent of v with storage location $P(v)$. Figure 1 contains a single phase of procedure dynamic contraction.

The procedure must detect when $Arg(v)$ equals 0 or 1. If the number of arguments per vertex is bounded this can be tested in constant time using the processor assigned to vertex v. In the case when the number of arguments is unbounded we can assign a

Figure 1. A dynamic contraction phase.

Procedure Dynamic Tree Contraction:
In Parallel for all $v \in V - \{r\}$ **do**
1) **If** $Arg(v) = 0$ **then** mark $P(v)$ and delete v;
2) **If** $Arg(v) = Arg(vertex(P(v))) = 1$ **then** $P(v) \leftarrow P(vertex(P(v)))$.
od

processor to each argument still using at most $O(n)$ processors since the total number of arguments is $n - 1$. These processors assigned to the arguments of v can test whether $Arg(v) = 0$ by having each processor Q without an argument perform a concurrent write of its index into some memory location m_v of v. $Arg(v) > 0$ if and only if some index is written into this memory location. To further test if $Arg(v) > 1$, we have each processor Q with no argument read m_v and if the value is not the index of Q then Q again writes its index in m_v. Thus, $Arg(v) > 1$ if the value of m_v changes on the second write. In Section 6 we show that procedures that takes at most $O(\log k)$ to test $Arg(v)$ equal 0 or 1 will still give an overall running time of $O(\log n)$, where k is the number of arguments of v.

The procedure implements the RAKE in the straightforward way, while the operation COMPRESS is implemented by pointer jumping. In line (2) of the procedure each vertex in a chain adjusts its pointer P, which was initially pointing at its parent, to point at its grandparent.

More intuition for the procedure dynamic contraction can be gained by seeing it applied to expression evaluation over $\{+, \times\}$. If $Arg(v) = 0$ then v "knows" its value and passes it on to its parent. We can test if $Arg(v) = 0$ or $Arg(v) = 1$ in constant time using concurrent reads and writes. If v and $P(v)$ are functions of one remaining argument we will view them as linear functions of their argument. We store these functions in common memory indexed by the corresponding vertex. Thus v reads the linear functions of $P(v)$, composes it with its own function, and adjusts its pointer to $P\{vertex[P(v)]\}$. It follows that this correctly computes the value of the expression. We next analyze the number of applications of dynamic contraction.

THEOREM 3.1. *The number of applications of dynamic tree contraction needed to reduce a tree of n vertices to its root is bounded above by the number for CONTRACT.*

Proof. Observe that every maximal chain, after dynamic tree contraction, decomposes into two chains, one *essential chain* corresponding to COMPRESS and an unnecessary chain that is out of phase. This second chain has a leaf that is unevaluated. For the purpose of analysis we can discard the second chain from the analysis since it will never be evaluated. Thus, a single phase of dynamic tree contraction is just CONTRACT, after discarding the unevaluatable chains. It is important to point out that dynamic tree contraction is slightly faster than CONTRACT since it does not test if the only child of a vertex is a leaf or not. Thus, some pointer jumping occurs in dynamic tree contraction that does not occur in CONTRACT. We used the more conservative contract in CONTRACT since we felt that for many applications a vertex with an only child will use the time at this stage to evaluate itself rather than pointer jumping. □

Note that many vertices are not evaluated, that is, for many vertices v the value $Arg(v)$ is never set to 0 during any stage of dynamic tree contraction. We will define a new procedure dynamic tree expansion that will allow the evaluation of all vertices, i.e., each vertex will eventually have all its arguments after completion of the procedure. We modify dynamic tree contraction so that each vertex keeps a push-down store $Store_v$ of all the previous values of $P(v)$. Here we add line (0) at the start of the block inside the *do* and *od* of dynamic tree contraction.

0) Push on $Store_v$ value $P(v)$.

We now apply dynamic tree contraction until the root r has all its arguments. Next we apply procedure dynamic tree expansion given in Figure 2 until all vertices have all their arguments.

We must show that after successive applications of dynamic tree expansion all vertices have their arguments. As in the proof of Theorem 3.1 we can discard those chains that have a leaf that will not be evaluated. The proof is by induction on the trees with only

Figure 2. A dynamic expansion phase.

Procedure Dynamic Tree Expansion:
  In Parallel for all $v \in V - \{r\}$ **do**
    1) $P(v) \leftarrow Pop(Store_v)$:
    2) **if** $Arg(v) = 0$ **then** mark $P(v)$.
  od

essential chains, as defined in the proof of the previous theorem, starting from the trivial tree consisting of a singleton vertex r and finishing with the original tree T, say, $\{r\} = T_1, \ldots, T_k = T$. Now every vertex in T_{i+1} is either a leaf in which case we know its value or this vertex is missing one argument that is the value of a vertex in T_i. In the latter case this value will be supplied in one application of dynamic tree expansion. This gives the following theorem.

THEOREM 3.2. *At most* $\lfloor \log_{5/4} n \rfloor$ *applications of dynamic tree contraction and* $\lfloor \log_{5/4} n \rfloor$ *applications of dynamic tree expansion are needed to mark all the vertices.*

3.2. Randomized Tree Contraction and Expansion

We next describe a randomized version of CONTRACT. This algorithm has the disadvantage that it needs access to many random numbers but it has the advantages that (1) in many cases, it will only use about half as many function evaluations and (2) it can be modified into an algorithm that up to constant factors uses an optimal number $O(n/\log n)$ of processors and still runs in time $O(\log n)$. We describe the algorithm in procedure form (see Figure 3).

The rest of this section contains a probablistic analysis of the procedure randomized contract. We believe that good analysis of this procedure with attention to constants is important. We first show that roughly 1/5 of the vertices are deleted with probability at least 1/2. We use this bounded to show that randomized contract will reduce a tree to a single vertex in $O(\log n)$ time with high probability. For the processor efficient randomized contraction algorithms presented in Section 4.4 we need that randomized contraction deletes a constant proportion of the vertices with high

Figure 3. A RANDOMIZED CONTRACT phase.

```
Procedure RANDOMIZED CONTRACT:
    In Parallel for all v ∈ V − {r} which have not been deleted do
        1) If Arg(v) = 0 then mark P(v) and delete v;
        2) If Arg(v) = 1 then Randomly assign M or F to Sex(v);
            3) If Sex(v) = F and Sex(P(v)) = M then do
                a) Push on Store_v value P(v);
                b) P(v) ← P(P(v));
                c) delete vertex(P(v)).
            od
    od
```

probability for n large. Thus, we show that randomized contract deletes at least $n/32$ vertices with probability of failure at most $1/n$. Note that if we are not concerned about constants then the second analysis would suffice for both applications.

The analysis will follow arguments similar to those used in the proof of Theorem 2.1. Here we partition the vertex set V into Rake(V) and Compress(V) as defined in that proof. Again by similar argument step (1) of RANDOMIZED CONTRACT will delete at least $1/5$ of the vertices in Rake(V). Steps (2) and (3) of randomized CONTRACT are called *Randomized Pointer Jumping*. The expected number of vertices of Compress(V) that are deleted in step (3c) is $m/4$ where $m = |Compress(V)|$. We cannot directly conclude that the median is also $m/4$. Recall, that the *median* of a random variable X is the maximum real number $\mu(X)$ such that $Prob[X \leqslant \mu(X)] \leqslant 1/2$. We can lower bound the median using the expected number and the variance of the number of vertices deleted. Since the number of deleted vertices in each maximal chain is mutually independent, the number of deleted vertices is the sum of independent random variables, one for each maximal chain. Let $C_1, \ldots, C_k$ be a list of maximal chains in T where C_i is a chain of length $m_i + 1$. Thus, m_i of the vertices of C_i are members of the set Compress(V). Let the number of deleted vertices in the chain C_i after one application of RANDOMIZED CONTRACT be the random variable $MATE_{m_i}$. If $m = |Compress(V)|$ then the random variable that is the number of deleted vertices in one phase will be $X = MATE_{m_1} + \cdots + MATE_{m_k}$ where k is the number of maximal chains. Thus, the expected value of X is $E(X) = m/4$. By Lemma 7.1 the variance for one chain is $(m_i + 2)/16$. Thus, the variance for X is $\sum_{i=1}^{k} (m_i + 2)/16 = (m + 2k)/16$. The variance is maximized when each $m_i = 1$. In this case the variance is $Var(X) = 3m/16$. The Chebyshev's inequality gives the following estimate for the median of X, $\mu(X)$ (see [L], p. 244).

LEMMA 3.3. $|\mu(X) - E(X)| \leqslant \sqrt{2\,Var(X)}$.

Thus $\mu(X) \geqslant E(X) - \sqrt{2\,Var(X)}$. In our case this gives $\mu(X) \geqslant m/4 - \sqrt{3m/8}$. Therefore for sufficiently large m, $m \geqslant 150$, $\mu(X) \geqslant m/5$. After some simple computer calculations we conjecture that $\mu(X) \geqslant m/5$ for $m \geqslant 15$ (see Section 7).

THEOREM 3.4. *RANDOMIZED CONTRACT deletes at least* $n/5 - 150$ *vertices with probability at least* $1/2$.

Proof. Let T be the tree input to randomized contraction and $m = |Compress(V)|$. Thus, $n - m = |Rake(v)|$. We know that at least $(n - m)/5$ vertices in $Rake(v)$ are deleted in every phase. We know from the last lemma that for $m \geqslant 150$ at least $m/5$ of the vertices in Compress(V) are also deleted with probability 1/2. Thus, $m/5 - 150$ of the vertices in $Compress(V)$ are deleted with probability 1/2. Therefore the total deleted is at least $(n - m)/5 + m/5 - 150 = n/5 - 150$.

Let S_n be the number of successes in n independent trials with probability p of success on each trial. We shall need one major fact about the binomial random variable S_n – the probability of being more than any fixed constant factor from the expected value is exponentially small. This fact was observed by Uspensky [U] (see [JK]). These bounds are commonly known as Chernoff bounds [C]. We shall use the following simply stated bounds [AV].

THEOREM 3.5. *For any n, p, ε with $0 \leqslant p \leqslant 1$, $0 \leqslant \varepsilon \leqslant 1$:*

$$Prob(S_n \leqslant \lfloor (1 - \varepsilon)np \rfloor) \leqslant e^{-\varepsilon^2 np/2}$$

and

$$Prob(S_n \geqslant \lceil (1 + \varepsilon)np \rceil) \leqslant e^{-\varepsilon^2 np/3}.$$

We use these bounds to show:

THEOREM 3.6. *After $\lceil 12.5 \log n \rceil + 150$ applications of RANDOMIZED CONTRACT a tree of n vertices will be reduced to a single vertex with probability of failure at most $1/n$.*

Proof. We show that after $k = \lceil 12.5 \log n \rceil$ applications of RANDOMIZED CONTRACT a tree of size n is reduced to a tree of size 150. Since randomized contract always removes at least the leaves of a tree the tree of size 150 will take at most 150 more steps. We say a given application of randomized contract is a *success* if it deletes $n/5 - 150$ vertices from a tree of size n and a *failure* otherwise. If after k applications the tree has not been reduced to one of size 150 then we must have had less than $\lceil \log_{5/4} n \rceil$ successes. In Lemma 3.4 we showed that probability of success was at least 1/2 independent of the tree. Thus the probability that the tree has more than 150 vertices after k application is bounded by $Prob(S_k \leqslant \lfloor \log_{5/4} n \rfloor)$ where $p = 1/2$. We use the first inequality from Theorem 3.5 to bound this probability. We set $n = k$, $\varepsilon = 1/2$, and $p = 1/2$

and check that $\lfloor \log_{5/4} n \rfloor \leqslant \lfloor (1-\varepsilon)kp \rfloor$. The last inequality is a straightforward calculation. Thus the probability of failure after k applications is at most $e^{-\varepsilon^2 kp/2}$. To get a probability of error at most $1/n$ we must just see that $\varepsilon^2 kp/2 \geqslant \ln n$. That is $k \geqslant 16 \ln n$, but $16 \ln n \leqslant 12 \log n \leqslant k$. □

We next show that RANDOMIZED CONTRACT will delete at least $n/32$ vertices with only vanishingly small probability of failure.

THEOREM 3.7. *One phase of RANDOMIZED CONTRACT for any $n \geqslant 180$ will delete at least $n/32$ vertices with a probability of failure less than $1/n$.*

Proof. Let n be the number of vertices in a tree T and m be the number of vertices in Compress(T). If $m \leqslant 27n/32$ then $n - m \geqslant 5n/32$ vertices are in Rake(T) and therefore at least $1/5(5n/32) = n/32$ of them are deleted by RAKE. In this case $n/32$ of the vertices are deleted by RAKE alone without considering vertices deleted by COMPRESS. Thus, we may assume that $m > 27n/32$. It will suffice to show that $m/32$ of the vertices in Compress(T) are deleted by RANDOMIZED CONTRACT with small probability of failure. Let $I \subset$ Compress(V) be a maximum subset of vertices such that no vertex in I is a parent of another vertex in I, i.e. I is a maximal independent set. Now each vertex in I is deleted independently with probability $1/4$. Since the induced graph on Compress(T) is a forest, the number of vertices in $|I| \geqslant \lceil m/2 \rceil$. Thus, the number of vertices deleted is bounded below by the binomial random variable $S_{\lceil m/2 \rceil}$ where $p = 1/4$. The probability of failure is bounded by

$$\text{Prob}\,(S_{\lceil m/2 \rceil} \leqslant \lfloor m/32 \rfloor) \leqslant \text{Prob}\,(S_{\lceil m/2 \rceil} \leqslant \lfloor (1-\varepsilon)\lceil m/2 \rceil/4 \rfloor),$$

where $\varepsilon = 3/4$. Using the Chernoff bounds from Theorem 3.5 this probability at most

$$\leqslant e^{-\varepsilon^2 \lceil m/2 \rceil/8} \leqslant e^{-\varepsilon^2 m/16}.$$

Using the hypothesis that $m \geqslant 27n/32$ and $\varepsilon = 3/4$ we get the above probability

$$\leqslant e^{-(3/4)^2(27/32)n/16} = e^{-(3^5/2^{13})n} \leqslant e^{-n/34}.$$

For $n \geqslant 180$ we have that $e^{-n/34} \leqslant 1/n$. □

4. AN OPTIMAL RANDOMIZED TREE EVALUATION ALGORITHM

4.1. Improving the Processor Count by Load Balancing

In this section we show how to implement RANDOMIZED CONTRACT on a tree T with n vertices so that T is reduced to its root in $O(\log n)$ time using $O(n/\log n)$ processors. We will contract T to a tree T' of size $n/\log n$ at which point we will have a tree small enough so there is a processor for every vertex of T' and can use one the deterministic algorithm from the last section. The important difference in the reduction is that we will be operating on an array of n vertices using only $o(n)$ processors as opposed to one processor for each pointer value. We consider pointers to be either *dead* or *alive*. If all pointers of the array are alive and we have p processors then we simply assign intervals of pointer values of size $\lceil n/p \rceil$ to a single processor.

If the live pointers are interspersed with dead pointers then the time required for some particular processor to finish its tasks may be much longer than the expected or average time. We give a method of balancing the work load using randomization. We consider the processors to be numbered consecutively. In general if A is an algorithm originally specified using p processors but only p' are available we will assume that A is implemented by assigning each distinct interval of $\lceil p/p' \rceil$ virtual processors to one actual processor.

Note that by Theorem 3.7 after each phase of randomized contract with very high probability at least 1/32th of the processors are assigned to dead pointers. Thus, after $O[\log(\log n)]$ phases we will have only $n/\log n$ active processors. One can assign active tasks to an initial sequence of processors by computing all prefix sums as follows.

Let $s_1 \ldots s_n$ be a sequence of zeros and ones where $s_i = 1$ if processor i is active and $s_i = 0$ otherwise, and $a_k = \sum_{i=1}^{k} s_i$. We now assign the task of processor i to processor a_i. It is well known, see [LF]:

LEMMA 4.1. *All prefix sums of a string of length n can be computed in* $O(\log n)$ *time using* $n/\log n$ *processors.*

This motivates a simple randomized tree evaluation algorithm using $O(n \log(\log n)/\log n)$ processors and $O(\log n)$ time (see Figure 4).

Figure 4. A Randomized tree evaluation (simple form).

Procedure Randomized Tree Evaluation (Simple form):
1) $p \leftarrow \lceil n \log(\log n)/\log n \rceil$; $k \leftarrow 1$;
2) **While** $k \leq c(\log(\log n))$ **do**
   $T \leftarrow$ Randomized Contraction(T); (using p processors) (*)
3) Using all prefix sums calculation assign the active tasks to an initial sequence of processors;
4) **While** $|T| > 1$ **do** $T \leftarrow$ RANDOMIZED CONTRACT(T)

To see that it works in $O(\log n)$ time we use Theorem 3.7. Note that for some constant c and large enough n, step (2) will reduce T to a tree on $\lceil n/\log n \rceil$ vertices with probability of failure $\leqslant 1/n$. Now each execution of (*) will take $O[\log n/\log(\log n)]$ time. Thus, step (2) requires $O(\log n)$ time. By Lemma 4.1 step (3) takes only $O(\log n)$ time. By the first remark and large enough c we have $|T| \leqslant n/\log n$. Thus, step (4) will only take $O(\log n)$ time with probability of failure $\leqslant 1/n$.

Thus, the simple form of randomized tree evaluation reduces the processor count to $O[n \log(\log n)/\log n]$, by only "load balancing" once. To remove the last $\log(\log n)$ factor we will load balance between each application of (*). The goal will be to partially balance the load as opposed to performing the balancing exactly. We do the partial balancing by first randomly permuting the tasks and next partially balancing the almost random string of tasks.

4.2. Generating a Random Permutation

In this section we give a processor efficient algorithm to generate random permutations. Another algorithm appears in [R2]. In particular we show:

THEOREM 4.2. *There exists a randomized PRAM algorithm that generates random permutations of n cells using $O(\log n)$ time, $O(n/\log n)$ processors, and probability of failure is at most $1/n$.*

The idea behind the algorithm is extremely simple. We shall randomly assign the n cells among $2n$ cells. We call the assigned position an *accommodation*. Next we remove the unused cells using prefix calculations as described in the previous section. To get the original assignment of the n cells into $2n$ we will require each of the $n/\log n$ processors to be responsible for finding accommodations for

$\log n$ cells. Each processor starts at the beginning of its list of cells and chooses a random accommodation. The processor will find an accommodation for the cell with probability at least $1/2$. Thus, the expected completion time for each processor is at most $2 \log n$. We allow each processor $12\lceil \log n \rceil$ trials. If after this many trials, it has not found accommodations for all its cells the process as a whole is aborted using the concurrent write ability.

LEMMA 4.3. *The probability that the above procedure aborts is at most* $1/n$.

Proof. Let Y be a random variable corresponding to the number of accommodations found after $t = 12(\lceil \log n \rceil)$ trials. Since each trial finds an accommodation with probability at least $1/2$ the random variable Y is bounded from above by a binomial random variable X with $p = 1/2$ on t trials. That is $\text{Prob}(Y \leqslant x) \leqslant \text{Prob}(X \leqslant x)$ for all x.

Here we use the Chernoff bound:

$$\text{Prob}(X \leqslant \lfloor (1 - \varepsilon)pt \rfloor) \leqslant e^{-\varepsilon^2 tp/2}.$$

Setting $\varepsilon = 5/6$, $p = 1/2$, and $t = 12\lceil \log n \rceil$ we get

$$\text{Prob}(X \leqslant \lceil \log n \rceil) \leqslant e^{-(25/12)\lceil \log n \rceil} \leqslant e^{-2 \log n} \leqslant 1/n^2.$$

Thus, the probability of failure for any given processor is at most $1/n^2$. Therefore, failure probability as a whole is at most $1/n$. □

4.3. Removing a Constant Proportion of Zeros from a Random String

Let $\sigma = s_1 \ldots s_n$ be a random binary string where each s_i is an independent random variable that takes the value one with probability p and zero with probability $q = 1 - p$. We view σ as a sequence of live and dead cells where the ith cell is alive if $s_i = 1$ and dead if $s_i = 0$. One can remove all dead cells by computing all partial sums.

Thus, all dead cells can be removed in $O(\log n)$ time using $O(n/\log n)$ processors. We need a faster algorithm that uses only $O(\log(\log n))$ time and $O(n/\log(\log n))$ processors. But we require

only that the algorithm remove a constant proportion of the dead cells in a random string.

We shall say that an algorithm on an input string σ of length n *discards* k zeros if it maps the nonzero elements in a one-to-one way into a new string of length at most $n - k$.

THEOREM 4.4. *There exists a PRAM algorithm DISCARD ZEROS using* $O(\log(\log n))$ *time and* $O(n/\log(\log n))$ *processors that, for at least* $1 - 1/n$ *of the random strings* σ *of length n discards at least* $\lceil qn/2 \rceil$ *zeros,* $p = 1 - q$ *fixed and n sufficiently large.*

Proof. We partition n into intervals of size $m = \lceil c(\ln n) \rceil$ plus one last interval of size $\leqslant m$. We fix c as a function of q later in the proof. Each interval will be given $k = \lfloor (p + q/2)m - 1 \rfloor$ consecutive storage locations in which to store its live cells. We assign $O[m/\log(\log n)]$ processors to each interval. In $O(\log m)$ time, using the classical prefix sums algorithm, these processors place the live cells in their interval. If any interval has more live cells than storage locations then the process as a whole is aborted using concurrent write. The algorithm has thus failed on this input.

We first check that if the algorithm terminates then it has in fact it has discarded $\lceil q/2n \rceil$ zeros. The total space used for storing the ones is $\lceil n/m \rceil \lfloor (p + q/2)m - 1 \rfloor$, which is less than or equal to

$$(n/m + 1)[(p + q/2)m - 1]$$

$$= (p + q/2)n + (p + q/2)m - n/m - 1.$$

For $n \geqslant (p + q/2)m^2$ this sum is less than $\lfloor (p + q/2)n \rfloor$. Thus, we have discarded $\lceil q/2n \rceil$ zeros.

Before we show that the algorithm fails only on a vanishingly small fraction of the strings we analyze the number of processors and the time uses. Sincce there are $\lceil n/m \rceil$ intervals each using $O(m/\log(\log n))$ processors the total number of processors used in $O(n/\log(\log n))$. Since each interval can be packed in parallel, the total time (besides computing the parameters m and k) will just be the cost of computing all the prefix sums for a string of length m, which is $O(\log m) = O(\log(\log n))$.

The procedure fails on some interval if the number of zeros in that interval is less than $\lceil q/2m \rceil$. It will suffice to show that the probability of fewere than $\lfloor q/2m + 1 \rfloor$ zeros in an interval is small.

Note that $q/2m + 1 = (1 - \varepsilon)qm$ for $\varepsilon = 1/2 - 1/(qm)$ and for m large $\varepsilon > 0$. To analyze the probability of failure we use Chernoff bounds from Theorem 3.5. Let S_m be a binomial random variable with parameters m, q. We have the following bound on the probability of failure for some interval:

$$\text{Prob}\,(S_m \leqslant \lfloor (1 - \varepsilon)mq \rfloor) \leqslant e^{-\varepsilon^2 mp/2}.$$

Using our values of ε and m, and setting $c = 9/q$ we get

$$e^{-[1/2 - 1/(qm)]mq/2} = e^{-(mq/2 - 1/2)} = e^{-2\ln n - (\ln n - 1/2)} \leqslant 1/n^2.$$

The last inequality follows for $n \geqslant 2$. Now the probability of failure on any interval is upper bounded by $(n/m)1/n^2 = 1/mn$. Since $m \geqslant 2$ we get that failure occurs less than $1/n$ of the time. □

THEOREM 4.5. *There exists a PRAM algorithm using* $O(\log(\log n))$ *time and* $O(n/\log(\log n))$ *processors that for at least* $1 - 1/n$ *of the strings with b zeros discards at least* $\min\{b/2, n/3\}$ *zeros.*

Proof. To prove the theorem we use the algorithm from the proof of the previous theorem with $p = (n - b)/n$. The analysis of failure for the previous theorem reduces to Chernoff bounds for tails of a binomial random variable with parameters m, p. In this case the random variable is hypergeometric with parameters n, m, $n - b$. Hoeffding [H], Theorem 4, has shown that the moments of a hypergeometric are always bound by the moments of a binomial with the same expected value. Thus, the Chernoff bounds in Theorem 3.5 can be applied directly to hypergeometric distributions. Thus the arguments used in the proof of Theorem 4.4 apply directly to this case giving an error bound of $1/n$. □

4.4. Randomized Tree Evaluation Using $O(n/\log n)$ Processors

We are now ready to describe our optimal randomized tree evaluation algorithm. The procedure is presented in Figure 5. Routine (a) generates for each i an upper bound x_i on the size of the work space at the ith stage of routine (c). The routine (b) generates in parallel all the permutations that will be needed in routine (c). We generate all the permutations at once to ensure $0(\log n)$ time.

Figure 5. An optimal randomized tree evaluation algorithm.

Procedure Randomized Tree Evaluation:
  Set $x_1 \leftarrow n; \alpha \leftarrow 31/32$; $k \leftarrow 1; i \leftarrow 1; T_1 \leftarrow T$;
  While $x_i \geq n/\log n$ **do** (a)
    1) $x_{i+1} \leftarrow \lceil \alpha x_i \rceil$;
    2) $i \leftarrow i + 1$
  In Parallel Generate random permutations $\pi_1, \ldots, \pi_i$ (b)
    of sizes $x_1, \ldots, x_i$, respectively
  While $k < i$ **do** (c)
    1) $T_{k+1} \leftarrow$ RANDOMIZED CONTRACT(T_k), using p processors;
    2) Permute the pointers of T_{k+1} using π_{k+1};
    3) Apply DISCARD ZEROS to the list of pointers T_{k+1}
      returning at most x_{k+1} pointers;
    4) $k \leftarrow k + 1$.
  While $|T| > 1$ **do** (d)
    $T \leftarrow$ Randomized Contraction(T)
      (using a distinct processor at each vertex)

Routine (*c*) step (1) for each k contracts T_k to T_{k+1} generating at least $x_k/16$ dead pointers. After randomly permuting the pointers, step (2), step (3) discards at least $x_k/32$ dead pointers. When routine (*d*) is implemented, T will be stored in an array of pointers of size at most $0(n/\log n)$. Since no routine will be implemented more than $0(\log n)$ times we need only make sure that the probability of aborting at each step is $\leqslant 1/cn \log n$ for some constant c. These bounds follow from the preceding theorems and the fact that the error can be decreased to $1/n^2$ by simply running an algorithm twice.

We discuss each of the four routines: (*a*), (*b*), (*c*), and (*d*). Routine (*a*) is deterministic and thus always works. Routine (*b*) generates all $O(\log n)$ permutations needed by the algorithm. The important fact to note is that the sum of their sizes is $O(n)$. Thus, we can generate each π_i in $\log n$ time using at most $O(x_i/\log n)$ processors (see Theorem 4.2) all with probability of failure at most $1/n$. Therefore routine (*b*) uses $O(\log n)$ time and $n/\log n$ processors. The analysis of routine (*c*) is slightly more complicated. By Theorem 3.7 RANDOMIZED CONTRACT will fail with probability at most $1/n$ for sufficiently large n. By Theorem 4.5 DISCARD ZEROS will fail with probability at most $1/n$ also. Step (1) will take $O(x_k/n \log n)$ time using $n/\log n$ processors. Since x_k/n is approximately α^k the time taken by (1) is geometrically decreasing in k. Therefore the total time used by (1) over all values of k is $O(\log n)$. This same analysis also applies to (2). Using

Theorem 4.5, the procedure DISCARD ZEROS can be speeded only by using more processors to a minimum time $O(\log(\log n))$. Thus, we must check that $x_i/n \log n \geqslant \log(\log n)$. But $x_i \geqslant 31/32n$. Thus, for n large step (3) also uses total time at most $O(\log n)$. Finally routine (d) will complete in $O(\log n)$ time by Theorem 3.6 with failure probability at most $1/n$. Thus, it follows that RANDOMIZED TREE EVALUATION will fail with probability at most $1/n$.

Using the parallel tree expansion ideas in Theorem 3.2 we get:

THEOREM 4.6. *There exists a 0-sided randomized algorithm that marks all vertices of a tree in $O(\log n)$ time using $O(n/\log n)$ processors.*

For deterministic dynamic tree expansion, we had enough processors so that all the trees $T_1, \ldots, T_k$ computed during dynamic tree contraction can be stored on the processors local memory using a pushdown store. Here we have fewer processors so we shall simply store the tree in common memory with back pointers from vertices in tree T_i to corresponding vertices in T_{i+1}. Since the size of the trees is decreasing geometrically the total storage is at most linear.

5. APPLICATIONS OF DYNAMIC TREE CONTRACTION

5.1. Arithmetic Expression Evaluation

Let T be a tree with vertex set V and root r. We assume each leaf is initially assigned a value $C(v)$, and each internal vertex v, with children $u_1, \ldots, u_k$, has a label $L(v)[u_1, \ldots, u_k]$ that is assumed to be of the form $\theta(u_1, \ldots, u_k)$ where $\theta \in \{+, -, \times, \div\}$. A bottom-up approach for expression evaluation is to substitute $L(u_i)$ into $L(v)[u_1, \ldots, u_k]$ for each child u_i that is a leaf, and then delete u_i. This method however requires time $\Omega(n)$ in the worse case. The results of Brent imply we can do expression evaluation in $O(\log n)$ time if we can preprocess the expression [6]; however $\Omega(\log n)^2$ time seems to be required if the expression is to be evaluated dynamically (i.e., on line).

THEOREM 5.1. *Dynamic arithmetic expression evaluation can be done in $O(\log n)$ time using $O(n)$ processors deterministically and only $O(n/\log n)$ processors using a 0-sided randomized procedure.*

Proof. We shall assume that the number of arguments at a vertex is at most 2. If not we assume that in $O(\log n)$ time we can convert it into such a tree. As in Brent we shall perform only one division at the end.

The values stored or manipulated will be sums, products, and differences of the initial leaf values $C(v)$. The value returned will be a ratio of these elements. The operations $\{+, -, \times, \div\}$ will have their usual interpretations, e.g., $a/b + c/d = (ad + bc)/bd$. The other main item we need is a way to represent elements from a class of many functions that is closed under composition. Here we will use ratios of linear functions of the form, $(ax + b)/(cx + d)$. We must verify that they are closed under composition:

$$\frac{a'(au + b)/(cu + d) + b'}{c'(au + b)/(cu + d) + d'} = \frac{a''u + b''}{c''u + d''}.$$

□

5.2. Arithmetic Subexpression Evaluation

By running procedure randomized tree evaluation "backward" (Figure 5) as we did in the deterministic case (Figure 3) we get:

THEOREM 5.2. *All subexpressions can be computed in the time and processor bounds in Theorem 5.1.*

A special case of computing all subexpressions is the linked-list ranking problem. Here we have a linked-list and we would like to compute the position in the list of all elements (see [V]).

COROLLARY 5.3. *General a linked-list of size n, the position of each element in the list can be computed with a* 0*-sided randomized algorithm in* $O(\log n)$ *time using* $O(n/\log n)$ *processors.*

6. PARALLEL TREE CONTRACTION FOR TREES OF UNBOUNDED DEGREE

Up until now we have assumed that the RAKE operation could be performed in unit time. For many applications this is not the case. As we shall see in the companion paper [MR2], the RAKE operation for certain applications may be considerably more complicated than just deletion. As an example, we may need to sort

the labels assigned to the leaves of a vertex before they can be "ranked." Here, the parallel time of raking the leaves of a vertex with k children is $O(\log k)$. If we require the algorithm in this example to finish a CONTRACT completely before it is allowed to start the next CONTRACT then it is not hard to construct examples where the total cost to reduce a tree to its root will be the cost for RAKE times the logarithm of the size. As an example of where the Rake operation is not constant time in Part 2 of this paper we consider the problem of finding canonical labels for trees; here the RAKE operation consists of sorting the labels of all siblings before the leaves are removed (see [MR2] for details). It is well-known how to sort in $O(\log^2 n)$ time. Thus, the naive analysis of this algorithm would be that it runs for $O(\log^2 n)$ time. We will improve the running time by a factor of $\log n$ below.

We modify parallel tree contraction so that for those parts of the tree where CONTRACT has already finished we implement a new round of CONTRACT, i.e., CONTRACT is run asynchronously. We shall assume that the time used to remove the leaves of a given vertex is only a function of the number of leaves at that vertex. We should point out that the synchronous and asynchronous versions of CONTRACT may return very different answers. In the case of computing canonical forms for trees by sorting leaves both the synchronous and asynchronous algorithms are correct. The asynchronous version will be faster.

Asynchronous Parallel Tree Contraction (APTC) can be described graph theoretically by viewing it as operating on trees with special leaves that we call *phantom leaves*. The algorithm APTC is run in stages. Initially the tree T has no phantom leaves. We apply the procedure CONTRACT to T obtaining the tree T'. If a given vertex $v \in T$ had $k \geqslant 2$ children that are leaves excluding any phantom leaves then we add a new phantom child w to $v \in T'$. Further, if the time required for APTC to delete these k children of v is t then the phantom vertex w will persist for t stages at which time it simply disappears. Note that a given vertex may have several phantom children and a vertex with a phantom child is not a leaf. The time to execute APTC is the number of stages it takes to reduce the tree to its root and all phantom leaves to disappear.

THEOREM 6.1. *If the cost to RAKE a vertex with k children is bounded by* $O(\log k)$ *then asynchronous tree contraction requires only* $O(\log n)$ *time.*

Proof. Suppose the time to RAKE a vertex with k children is bounded by $c \log k$ for $k \geqslant 2$ and RAKE for a single child can be performed in unit time. We shall analyze the time used by asynchronous parallel tree contraction by assigning weights to the vertices of the tree such that at any stage of the algorithm the weight of the tree reflects the progress made so far.

A *weighted tree* is a tree with weights assigned to the vertices. The weight of a tree is the sum of the weights of the vertices in the tree. In this application all vertices will have weight one except phantom leaves, which may have arbitrary real weights $\leqslant 1$. Thus, initially, the weight of the tree is the size of the tree.

We describe in more detail how weights are assigned to phantom leaves. Suppose the time required to rake the k nonphantom leaves of a vertex v is $f(k)$. There is a subtle point that is worth pointing out. Namely, if the time to rake a vertex with k leaves varies from vertex to vertex this may dramatically affect the way the tree contracts. Our analysis depends only on an upper estimate for the time to rake a vertex. We pick β, which is a function of k, such that $\beta^{f(k)-1}k = 1$ for $f(k) > 0$. Hence $\beta < 1$ for all $k \geqslant 2$. The constant β will be the rate at which the phantom leaf decays. We set the weight of the new phantom leaf w of v to βk. After each successive stage we will decrease the weight on w by a factor of β until the weight equals one. In the next stage we will simply delete the phantom leaf w. Thus, the phantom leaf w will exist for $f(k)$ stages at which time it will be deleted. Note that the weight of a phantom vertex is always $\geqslant 1$.

As in the proof of Theorem 2.1 we partition the vertices of T into two sets, *Ra* and *Com*. We claim that the weight of *Com* decreases by a factor of 1/2 at each stage while the weight of *Ra* decreases by a factor of at least $(4 + \beta)/5$ at each stage, where $\beta = \max\{\beta(k) | 1 \leqslant k \leqslant n\}$. Note that different phantom leaves decay at different rates. We have picked β to be the slowest such rate. The fact that *Com* decreases by 1/2 follows by noting that the vertices in *Com* are processed the same as in CONTRACT and their weights are all one. We next consider the case of *Ra*, $Ra = V_0 \cup V_2 \cup C_0 \cup C_2 \cup GC_0$ where V_0 is the set of leaves and phantom leaves. Since the weight on any vertex in V_0 is at least one and the weight of any vertex not in V_0 is 1 we see that that weight of V_0 is at least 1/5 of the weight of *Ra*. On the other hand the weight of V_0 decreases by at least β at each stage. Thus, the weight of *Ra* decreases by at least a factor of $4/5 + \beta/5$ at each stage.

This shows that the number of stages is bounded by $\log n$ base $5/(4+\beta)$. For a particular case of interest when $f(k) \leqslant c \log k$ for some constant c and $k \geqslant 2$ we see that β is bounded away from 1 for all n. This proves the theorem. □

7. THE RANDOM VARIABLE MATE

Let Σ_n be the space of all binary strings of length $n+1$ for $n \geqslant 1$. Let $MATE_n$ be a random variable defined on Σ_n where $MATE_n$ equals the number of 01 patterns in a string from Σ_n. Intuitively, 0 is a female and 1 is a male.

LEMMA 7.1. *The random variable $MATE_n$ has expected value $n/4$ and variance $(n+2)/16$.*

Proof. Let $s_0 \ldots s_n$ be a random binary string. Since the expected value of $MATE_2$ substring $s_i s_{i+1}$ is $1/4$ and there are n such substrings the expectation for $s_0 \ldots s_n$ must be $n/4$. Here we used the fact that expectations sum.

To compute the variance we consider a slightly different random variable with the same probability distribution. Let S_n be the binomial random variable on binary strings of length n with $p = 1/2$. We define a random variable X with $p = 1/2$ over the space of all zero–one strings of length $n+1$ as follows:

$$X(t_0 \ldots t_n) = \begin{cases} \lceil (S_n(t_1 \ldots t_n))/2 \rceil & \text{if } t_0 = 0 \\ \lfloor (S_n(t_1 \ldots t_n))/2 \rfloor & \text{if } t_0 = 1. \end{cases}$$

To see that X is simply a change of variables of $MATE_n$ consider the mapping from $s_0 \ldots s_n$ to $t_0 \ldots t_n$ defined by $t_0 \leftarrow s_0$ and inductively $t_i = 0$ iff $s_{i-1} = s_i$. One can see that this mapping is surjective and $X(t_0 \ldots t_n) = MATE_n(s_0 \ldots s_n)$. Thus, the expected value of X is $n/4$ and we need only compute the second moment of X, $E(X^2)$.

$$\begin{aligned} E(X^2) &= 1/2 \sum_{k=0}^{n} \{\lceil k/2 \rceil^2 \operatorname{Prob}(S_n = k) + \lfloor k/2 \rfloor^2 \operatorname{Prob}(S_n = k)\} \\ &= 1/2 \sum_{k \text{ odd}} [(k^2+1)/2] \operatorname{Prob}(S_n = k) \\ &\quad + 1/2 \sum_{k \text{ even}} (k^2/2) \operatorname{Prob}(S_n = k) \\ &= 1/4 \left[\sum_{k=0}^{n} k^2 \operatorname{Prob}(S_n = k) + \sum_{k \text{ odd}} \operatorname{Prob}(S_n = k) \right] \end{aligned}$$

The first term in the sum is just 1/4 of the second moment of S_n, which is $(n^2 + n)/4$. By a straightforward examination of Pascal's Triangle the second term equals 1/2, since the sum consists of every other term in a row of Pascal's triangle, which is equal to the sum of the row above it. Thus, $E(X^2) = (n^2 + n + 2)/16$. Therefore the $\text{var}(X) = E(X^2) - E^2(X) = (n + 2)/16$.

By similar arguments we get the following bound on $MATE_n$:

LEMMA 7.2. $\forall x$ *Prob* $(\lceil S_n/2 \rceil \leqslant x) \leqslant$ *Prob* $(MATE_n \leqslant x) \leqslant$ *Prob* $(\lfloor S_n/2 \rfloor \leqslant x)$.

One of the referees has noted that Lemma 7.1 follows by a rather straightforward induction based on covariances. We feel that our proof while slightly longer is instructive since it shows that the random variable $MATE_n$ is very closely related to a simple binomial random variable. We conjecture that the random variable $MATE = MATE_{n_1} + \cdots + MATE_{n_m}$ where $n = n_1 + \cdots + n_m$ has all its moments bounded by the moments of S_n for $p = 1/4$. If the conjecture is true the analysis of many of the theorems could be simplified and the constant improved. We hope that this section is of some help in settling this conjecture.

ACKNOWLEDGEMENTS

This work was supported in part by National Science Foundation Grant DCS-85-14961 and Air Force Office of Scientific Research AFOSR-82-0326 (to G.L.M.) and office of Naval Research Contracct N00014-80-C-0647 and National Science Foundation Grant DCR-85-03251 (to J.H.R.).

NOTES

1. Preliminary version of this paper appeared in Miller and Reif [MR1].

REFERENCES

[AV] D. Angluin and L. G. Valiant, Fast probabilistic algorithms for hamiltonian paths and matchings, *J. Computer System Sci.* (18): 155–193 (1979).

[BV] I. Bar-On and U. Vishkin, Optimal parallel generation of a computation tree form, *ACM Trans. Programming Languages Systems* 7(2): 348–357 (April 1985).

[B] R. P. Brent, The parallel evaluation of general arithmetic expressions, *J. Assoc. Computing Mach.* 21(2): 201–208 (April 1974).

[C] H. Chernoff, A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statistics* 23, 1952.

[F] Faith E. Fich, New bounds for parallel prefix circuits, in *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pp. 100–109, ACM, Boston, MA, April 1983.

[H] W. Hoeffding, Probability inequalities for sums of bounded random variables, *Amer. Statistical Assoc. J.* 13–30 (March 1963).

[JK] N. J. Johnson and S. Katz, *Discrete Distributions*. Houghton-Mifflin, Boston, MA, 1969.

[LF] Richard E. Ladner and Michael J. Fisher, Parallel prefix computation, *J. Assoc. Computing Mach.* 27(4): 831–838 (October 1980).

[L] M. Loeve, *Probability Theory*, Vol. 1, 4th ed., Springer, Berlin, 1977.

[M] Gary L. Miller, Finding small simple cycle separators for 2-connected planar graphs, *J. Computer System Sci.* 32(3): 265–279 (June 1986).

[MR1] Gary L. Miller and John H. Reif, Parallel tree contraction and its applications. In *26th Symposium on Foundations of Computer Science*, pp. 478–489, IEEE, Portland, Oregon, 1985.

[MR2] Gary L. Miller and John H. Reif, Parallel tree contraction. Part 2: Further applications, *SIAM J. Computing*, submitted.

[R1] John H. Reif, On the power of probabilistic choice in synchronous parallel computations, *SIAM J. Computing* 13(1): 46–56 (1984).

[R2] John R. Reif, An optimal parallel algorithm for integer sorting, in *26th Annual Symposium on Foundations of Computer Science*, pp. 496–504, ACM, 1985.

[SV] Y. Shiloach and U. Viskin, An $O(\log n)$ parallel connectivity algorithm. *J. Algorithms* 3: 57–67 (1982).

[U] J. Uspensky, *Introduction to Mathematical Probability*, 1st ed., McGraw-Hill, New York, 1937.

[V] U. Vishkin, Randomized speed-ups in parallel computation, in *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pp. 230–239, ACM Washington D.C., April 1984.

PRIVATE COINS VERSUS PUBLIC COINS IN INTERACTIVE PROOF SYSTEMS

Shafi Goldwasser and Michael Sipser

ABSTRACT

An interactive proof system is a method by which one party of unlimited resources, called the *prover*, can convince a party of limited resources, called the *verifier*, of the truth of a proposition. The verifier may toss coins, ask repeated questions of the prover, and run efficient tests upon the prover's responses before deciding whether to be convinced. This extends the familiar proof system implicit in the notion of NP in that there the verifier may not toss coins or speak, but only listen and verify. Interactive proof systems may not yield proof in the strict mathematical sense: the "proofs" are probabilistic with an exponentially small, though nonzero chance of error.

We consider two notions of interactive proof system. One, defined by Goldwasser, Micali, and Rackoff [GMR], permits the verifier a coin that can be tossed in *private*, i.e., a secret source of randomness.

Advances in Computing Research, Volume 5, pages 73–90.

ISBN: 0-89232-896-7

The second, due to Babai [B], requires that the outcome of the verifier's coin tosses be *public* and thus accessible to the prover.

Our main result is that these two systems are equivalent in power with respect to language recognition.

The notion of interactive proof system may be seen to yield a probabilistic analog to NP much as BPP is the probabilistic analog to P. We define the *probabilistic, nondeterministic, polynomial time Turing machine* and show that it is also equivalent in power to these systems.

1. INTRODUCTION

In this century, the notions of proof and computation have been formalized and understood. With the arrival of complexity theory, the notion of what is efficiently provable became of interest. The class NP captured this notion, containing those languages for which proofs of membership can be verified by a deterministic polynomial time Turing machine. We can view NP as a proof system consisting of two communicating Turing machines: the *prover* who guesses the proof and the polynomial time deterministic *verifier*, who checks the correctness of the proof.

Randomization has been recognized to be a fundamental ingredient in defining what is efficiently computable (e.g., RP, BPP, RNC). In this paper, we seek to understand how randomization affects the definition of what is efficiently provable.

A conventional deterministic NP verifier does not accept statistical evidence as a convincing argument, regardless of how overwhelming it may be. As a consequence, the kinds of languages contained in NP are precisely those whose proofs of membership can be fully put down in writing and shown to others. The verifier does not actively participate in the proof process or interact with the prover in any way. It suffices for the prover to speak and the verifier to listen.

Randomization and interaction are essential ingredients of two recent formalizations of the concept of an efficient proof system. One formalization is due to Babai [B] and the other to Goldwasser, Micali, and Rackoff [GMR]. Both definitions would collapse to NP if no coins were flipped.

1.1. Interactive Proof Systems

In defining what *interactive proof systems*, Goldwasser, Micali, and Rackoff's intent was to make as general a definition as

possible of what is provable to a probabilistic verifier willing to accept statistical evidence. Their broader goal was to define the concept of the "knowledge" communicated during a proof.

An *interactive proof system* consists of a prover with unlimited computation power and a probabilistic polynomial time verifier who receive a common input x. The prover and the verifier can exchange messages back and forth for a polynomial in the length of x number of times. There are no restrictions on how the verifier may use his coin tosses: he can toss coins, perform any polynomial time computation on them and send the outcome of the computation to the prover. In particular, he need not show the outcome of the coins to the prover.

The secrecy of the verifier's coin tosses seemed essential to certain examples of interactive proof systems. The most notable is a recent result of Goldreich, Micali, and Wigderson [GMW] showing an interactive proof system for the *graph nonisomorphism problem.* This is somewhat remarkable in light of the fact that graph nonisomorphism is not known to be in NP. We sketch this example in Section 2.1.

The interactive proof system (IP) defines a hierarchy of languages. Namely, L is in $IP[k]$ if there exists a k-move (k alternations of message exchanges between prover and verifier with the verifier sending the first message) interactive proof system such that for every input $x \in L$, the probability that the verifier accepts is greater than 2/3; and for every input x not in L, even against an optimal prover, the probability that the verifier accepts is less than 1/3.

1.2. Arthur–Merlin Games: An Interactive Proof System with a Public Coin

Babai's formalization of efficient proof system attempts to capture the smallest class of languages extending NP for which statistical proofs of membership exist. The primary motivation was to place the *matrix group nonmembership* and *matrix group order* problems in a complexity class "just above NP." His proof system, presented as a game, consists of a powerful prover (capable of optimal moves) called *Merlin*, and a probabilistic polynomial time verifier called *Arthur*, which receive a common input x. Merlin wins the game if he can make Arthur accept x. Arthur and Merlin alternate exchanging messages back and forth for at most a polynomial in the length of x times. At the end of the interaction, Arthur decides whether to accept or reject (i.e., whether Merlin won or lost).

The difference between the Arthur–Merlin proof system and the interactive proof system of [GMR] is in the restricted way that Arthur is allowed to use his coin tosses during the game. Arthur's moves consist merely of tossing coins and sending their outcomes to Merlin. Thus, the Arthur–Merlin game is a special case of an interactive proof system.

The Arthur–Merlin games define a hierarchy of complexity classes, in a manner similar to IP. We say L is in $AM[k]$ if there exists an Arthur–Merlin k-move game (i.e., k alternating message exchanges between Arthur and Merlin, Arthur sending first) such that for every input $x \in L$, the probability that Arthur accepts x is greater than 2/3; and for every input x not in L, the probability that an optimal Merlin wins is less than 1/3.

The elegant simplicity of the definition of the Arthur–Merlin game facilitates additional results. Babai showed that for every constant k, AM[k] collapses to AM[2]. This in turn is a subset of both Π_2^p and nonuniform NP. The relative power of proof systems with a bounded versus an unbounded number of exchanged messages remains an interesting open question.

In this paper we prove the equivalence of these two types of interactive proofs with respect to language recognition. As a consequence the above results extend to IP.

1.3. Our Result

Let Q denote a polynomial. Let $IP[Q]$ (and $AM[Q]$) denote those languages L for which there exists a Q-move interactive proof system (and Q-move Arthur–Merlin proof system, respectively). Namely, $Q(|x|)$ message exchanges between the prover and the verifier are allowed on input x.

In this paper, we show that for any polynomial Q,

$$IP[Q] \subseteq AM[Q + 2]$$

i.e., the GMR proof system is as powerful as the Babai proof system.

2. EXAMPLES AND RELATED WORK

2.1. An Example of an Interactive Proof System

Goldreich, Micali, and Wigderson [GMW] have recently demonstrated the following interactive proof system for the graph nonisomorphism problem.

Let $NONISO = \{(G_0, G_1)$ such that the graph G_1 is not isomorphic to the graph $G_0\}$.

THEOREM. (GMW): $NONISO \in$ IP.

Proof. The prover and verifier receive as input two n node graphs G_0 and G_1 on vertices V. The following steps 1 and 2 get executed n times in parallel.

Step 1. The verifer flips a fair coin to choose $c \in \{0, 1\}$ and a random permutation π of V. The verifier then computes $R = \pi(G_c)$ and sends R to the prover.
Step 2. The prover tells the verifier whether $c = 0$ or 1.
Final step. If the prover ever makes a mistake in Step 2 determining c the verifier rejects, otherwise he accepts.

If the two input graphs G_1 and G_2 are not isomorphic to each other, then there exists a prover who can distinguish the case that R is isomorphic to G_0 from the case that R is isomorphic to G_1, and thus can always tell correctly in Step 2 of the protocol whether $c = 0$ or $c = 1$, and make the verifier accept.

On the other hand, if G_0 is isomorphic to G_1, then by the randomness of the permutation π, R is as likely to be $\pi(G_0)$ as it is to be $\pi(G_1)$. The prover who does not know c will err in Step 2 of the protocol with probability 1/2. QED

Clearly, the secrecy of the coin is essential to this protocol.

One consequence of this result, combined with Babai's and ours, is that the graph nonisomorphism problem has polynomial-size, nondeterministic circuits.

Several other interactive proof systems for number theoretic problems and the matrix group membership problem appear in [GMR,B].

2.2. Related Work

The difference between IP and AM games is analogous to that between alternation [CKS] and alternation with partial information [R]. In the case of alternation both players play optimally subject to their knowledge and only the referee who determines the outcome is required to be polynomial time bounded. Condon and Ladner [CL] describe this connection in a general setting.

Other relevant results concerning interactive proof systems appear in [H] and [F]. Prior to our result, Johan Hastad [H] showed that the union $\bigcup_{k>0} IP[k]$ is contained in Σ_3. Paul Feldman [F] showed that the prover in an interactive proof system with a polynomial number of interactions need not be more powerful than a PSPACE machine. Boppana, Hastad, and Zachos [BHZ] showed that if co-NP $\subseteq$ AM then for any i, $\Sigma_i^p \subseteq AM$. This and our result show that the polynomial hierarchy collapses to Σ_2^p if graph isomorphism is NP-complete.

Fortnow and Sipser [FS] have shown that there is an oracle F such that co-NP$^F \not\subseteq$ IP^F. Aiello, Goldwasser, and Hastad [AGH] have shown that there is an oracle A such that IP^B = IP[2]^B.

Other work related to the study of randomized proof systems appear in [Pa] and [ZF]. In Papadimitriou's "Games Against Nature," the verifier is also a probabilistic polynomial time machine that flips coins and presents them to the prover, which is a capable of optimal moves. The difference is that the probability of convincing the verifier need not be bounded away from 1/2. This apparently affects the strength of the system as Papadimitriou's games are as powerful as PSPACE.

Furer and Zachos [ZF], in a work investigating the robustness of probabilistic complexity classes, introduce a framework of probabilistic existential and universal quantifiers and prove several combinatorial lemmas about them. The AM complexity classes can be formulated in terms of these special quantifiers.

2.3. Connections with Cryptography

An interactive proof system can be viewed as a model for proving the correctness of two party cryptographic protocols [GMR]. The prover and verifier in an interactive proof system model the two participants in a cryptographic protocol with one exception: the cryptographic prover is not all powerful, but a probabilistic polynomial time machine with a secret unknown to the verifier.

A key property of cryptographic protocols is the amount of "knowledge" released to the verifier during the execution of the protocol. Very informally, we say that a prover in an interactive proof system releases "zero-knowledge" if even a devious verifier can learn no more than the validity of the assertion being proved [GMR]. In [GMW] it has been shown that if one-way functions

exist then every language in NP has a zero-knowledge interactive proof system.

Using this and our transformation from private-coin to public-coin protocols, Ben-Or, et al. [BGGHKMR] has very recently shown that all languages in IP have zero-knowledge, public-coin protocols, given the existence of one-way functions. Our transformation does not preserve the zero-knowledge property. It is an open question whether there is such a transformation which does.

3. DEFINITIONS

We represent the *verifier* and the *prover* of an interactive proof system as two functions V and P.

DEFINITION. An *Interactive proof protocol* is given by two functions:

$$V: \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^* \cup \{\text{accept, reject}\}$$

$$P: \Sigma^* \to \Sigma^*$$

Let s_i denote the concatenation of i pairs of messages, $s_i = \#x_1\#y_1\#\cdots\#x_i\#y_i$. We write $V(w, r, s_i) = x_{i+1}$ to mean that V on input w, with random sequence r, and current message stream s produces next message x_{i+1}. We say $P(s_i \# x_{i+1}) = y_{i+1}$ to mean that P produces next message y_{i+1} given current message stream $s_i \# x_{i+1}$. The exchange of a single pair of messages is called a *round.*

For a given input w and random sequence r we say

$$(V^*P)(w, r) \text{ accepts}$$

if there exists a message stream $s = \#x_1\#y_1\#\cdots\#x_l\#y_l$ such that $V(w, r, s) = \text{accept}$, and for each $i < l$, $V(w, r, s_i) = x_{i+1}$ and $P(s_i \# x_{i+1}) = y_{i+1}$.

Let us assume for simplicity that there is a function l such that for inputs w of length n, V will only accept if the length of r is $l(n)$. Then we write

$$\Pr[(V^*P)(w) \text{ accepts}]$$

to mean $\Pr[(V^*P)(w, r) \text{ accepts}]$ for r chosen randomly from $\Sigma^{l(|w|)}$.

Further we let

$$\Pr[V(w) \text{ accepts}]$$

denote $\max_P \Pr[(V^*P)(w) \text{ accepts}]$.

Let the language of the verifier, $L(V) =$

$$\{w: \Pr[V(w) \text{ accepts}] > 1/2\}$$

Say V has error probability e if for all $w \in \Sigma^*$:

1. If $w \in L(V)$, $\Pr[V(w) \text{ accepts}] \geqslant 1 - e$
2. If $w \notin L(V)$, $\Pr[V(w) \text{ accepts}] \leqslant e$

For $W \subseteq \Sigma^*$, we say $W \in \text{IP}$ if there is a polynomial time verifier V with error probability $1/3$ accepting W. As we shall see later, the class IP is unaffected if we substitute e for $1/3$, where $2^{-\text{poly}(n)} \leqslant e \leqslant 1/2 - 2^{-\text{poly}(n)}$.

DEFINITION. An *interactive proof protocol with public coin* is defined as above with the following difference. The random input r is considered to be the concatenation of l strings $r = r_1 r_2 \ldots r_l$ where l is the number of rounds and V is restricted to produce r_i as its ith message, i.e., for $i \leqslant l$, $V(w, r, s_i) = r_i$ or *accept* or *reject*.

This notion is essentially identical to that of the Arthur–Merlin game defined by Babai in [B]. Following his terminology we say that for $W \subseteq \Sigma^*$, $W \in \text{AM(poly)}$ if $W \in \text{IP}$ as above and the interactive proof protocol uses a public coin. We refer to an Arthur–Merlin game as an A–M protocol.

For polynomial Q, say $W \in IP[Q(n)]$ if $W \in IP$ with a verifier that never sends more than $Q(n)$ messages for inputs of length n. Similarly define $AM[Q(n)]$.

4. THE EQUIVALENCE OF PUBLIC VS. PRIVATE COINS

4.1. Approximate Lower Bound Lemma

This lemma, an application of Carter–Wegman universal hashing [CW], plays a key role in our proof of equivalence. In its original form it appeared in Sipser [Si]. It was first applied to approximate lower bounds by Stockmeyer [St] and in Arthur–Merlin protocols by Babai [B]. The present simpler form of the lemma is due to Boppana (personal communication).

DEFINITION. Let D be a $k \times b$ Boolean matrix. The *linear function* $h_D: \Sigma^k \to \Sigma^b$ is given by $h_D(x) = xD$ using ordinary matrix multiplication modulo 2. A *random linear function* is obtained by selecting

the matrix D at random. If $H = \{h_1, \ldots, h_l\}$ is a collection of functions, $C \subseteq \Sigma^k$, and $D \subseteq \Sigma^b$ then $H(C)$ denotes $\bigcup h_i(C)$, and $H^{-1}(D)$ denotes $\bigcup h_i^{-1}(D)$. Let $|C|$ denote the cardinality of C.

LEMMA. Given m, $b > 0$ and $C \subseteq \Sigma^k$. Let $c \leqslant |C|/2^b$. Let h: $\Sigma^m \to \Sigma^b$ be a random linear function and z be an random member of Σ^b. Then:

$$\Pr[z \in h(C)] \geqslant c - c^2.$$

Proof. We may assume that $|C| \leqslant 2^b$ since decreasing $|C|$ to that size can only decrease $\Pr[z \in h(C)]$ while increasing $c - c^2$. The lemma requires the following two claims.

1. For $x \neq y \in \Sigma^m$, $\Pr[h(x) = h(y)] = 2^{-b}$.
2. $\Pr[z \in h(C)] \geqslant \Sigma_{x \in C} \Pr[z = h(x)]$
   $- \Sigma_{x \neq y \in C} \Pr[z = h(x) = h(y)]$.

The first claim follows because each bit of $h(x)$ has an independent 50% chance of agreeing with the corresponding bit of $h(y)$. The second follows from taking the first two terms of the inclusion–exclusion expansion. Since

$$\sum_{x \in C} \Pr[z \in h(x)] = |C|/2^b$$

and

$$\sum_{x \neq y \in C} \Pr[z = h(x) = h(y)] = \binom{|C|}{2} \Big/ 2^{2b} \qquad \text{(from claim 1)}$$

we see that

$$\Pr[z \in h(C)] \geqslant |C|/2^b - \binom{|C|}{2} \Big/ 2^{2b} \geqslant c - c^2 \text{ for } c \leqslant |C|/2^b \leqslant 1.$$

COROLLARY. Given m, b, $d \geqslant 0$ and $C \subseteq \Sigma^k$. Choose a random linear function $h: \Sigma^m \to \Sigma^b$ and a random $z \in \Sigma^b$.

1. If $|C| \geqslant 2^{b-2}$ then $\Pr[z \in h(C)] \geqslant 1/8$.
2. If $|C| \leqslant 2^b/d$ then $\Pr[z \in h(C)] \leqslant 1/d$.

Proof. Part 1 follows directly from the above lemma. Part 2 is straightforward.

COROLLARY. Given m, b, d, $l \geqslant 0$ and $C \subseteq \Sigma^m$. Choose random linear functions $H = \{h_1, \ldots, h_l\}$, $h_i: \Sigma^m \to \Sigma^b$ and $Z = \{z_1, \ldots, z_l\} \subseteq \Sigma^b$ then

1. If $|C| \geqslant 2^{b-2}$ then $\Pr[Z \cap H(C) \neq \emptyset] \geqslant 1 - (7/8)^l$.
2. If $|C| \leqslant 2^b/d$ then $\Pr[Z \cap H(C) \neq \emptyset] \leqslant l^2/d$.

Proof. Parts 1 and 2 follow directly from the previous lemma.

We use this lemma to obtain Arthur–Merlin protocols for showing an approximate lower bound on the size of sets. Let C be a set in which Arthur can verify membership, possibly with Merlin's help. Then Arthur picks random H and Z and Merlin attempts to respond with $x \in C$ such that some $x \in H^{-1}(z)$. If C is large then he will likely succeed and if C is small he will likely fail.

4.2. Main Theorem

THEOREM. $\mathrm{IP}[Q(n)] = \mathrm{AM}[Q(n) + 2]$ for any polynomial $Q(n)$.

An informal proof sketch. Let us focus on 1-round protocols. Assume V has an exponentially small error probability e, sends only messages of length m, and uses random sequences of length l. For each $x \in \Sigma^m$ let $\beta_x = \{r: V(r, w, \#) = x\}$. For every $y \in \Sigma^m$ let $\alpha_{xy} = \{r: r \in \beta_x \text{ and } V(r, w, \#x\#y) = \text{accept}\}$. Clearly, for each x, the optimal prover will select a y_x maximizing $|\alpha_{xy}|$. Let $\alpha_x = \alpha_{xy_x}$. Let $\alpha_0 = \bigcup_x \alpha_x$. Then $\Pr[V(w) \text{ accepts}] = |\alpha_0|/2^l$.

We next present the protocol by which A and M simulate V and P. M tries to convince A that $|\alpha_0| > e \cdot 2^l$ because this implies that $\Pr[V(w) \text{ accepts}] > e$ and hence ≈ 1. He does this by showing that there are "many" α_x's that are "large," where "many" $\times$ "large" $> e \cdot 2^l$. The tradeoff between "many" and "large" is governed by a parameter b sent by M to A.

More precisely, M first sends b to A. Then two approximate lower bound protocols ensue. The first convinces A that $|\{x: |\alpha_x| \geqslant e \cdot 2^l/2^b\}| \geqslant 2^b$. M produces an x in that set as per the approximate lower bound lemma. The second convinces A that x really is in that set as claimed, i.e., that $|\alpha_x| \geqslant e \cdot 2^l/2^b$.

For g-round protocols iterate the first approximate lower bound protocol to obtain $\alpha_0 \supseteq \alpha_1 \supseteq \cdots \supseteq \alpha_g$ where there are "many$_i$" ways to extend α_{i-1} to α_i and α_g is "large." Require that (Π "many$_i$") $\times$ "large" $\geqslant e \cdot 2^l$.

Full proof. Let $W \in \mathrm{IP}[Q(n)]$. We may assume, without loss of generality, that on inputs w of length n there are exactly $g(n) = Q(n)/2$ pairs of messages sent between V and P, these messages are exactly $m(n)$ long and the random input r to V is $l(n)$ long. Let $e(n)$ bound the error probability.

AMPLIFICATION LEMMA. Let $p(n)$ be a polynomial. Let V be a verifier that on inputs of length n sends a total of at most $g(n)$ messages, each of length $m(n)$, using $l(n)$ random bits, and with error probability at most 1/3. Then there is a V' such that $L(V) = L(V')$, with a total of at most $g(n)$ messages, each of length $\mathrm{O}[p(n)m(n)]$, using $\mathrm{O}[p(n)l(n)]$ random bits and with an error probability of at most $2^{-p(n)}$.

Proof. V' performs $\mathrm{O}[p(n)]$ independent parallel simulations of V and takes the majority vote of the outcomes. □

By this lemma we may assume

$$e(n) < l(n)^{-5g^2(n)}.$$

Further we may assume that $l(n) > \max[g(n), m(n), 50]$. We write g, m, e, l for $g(n)$, $m(n)$, $e(n)$, and $l(n)$ where n is understood.

We now describe the functions A and M simulating V and P, informally as two parties exchanging messages. The variables x_i and y_i represent messages sent by V and P, respectively. In essence, the idea is for A to use the random hash functions to force M to produce a generic run of the V, P protocol and then finally to prove that this run would likely cause V to accept. The numbers b_i that M produces roughly correspond to the log of the number of possible generic messages that V can make at round i.

Arthur's Protocol

Round 0:

A initially makes a null move and receives number b_1 from M. Go to round 1.

Round i $(1 \leqslant i \leqslant g)$:

So far A has received $b_1, \ldots, b_i$, and strings $x_1, \ldots, x_{i-1}, y_1, \ldots, y_{i-1}$ from M. Now A randomly selects l linear functions $H = \{h_1, \ldots, h_l\}$, $h_i: \Sigma^m \to \Sigma^{b_i}$ and l strings $Z = \{z_1, \ldots, z_l\} \subseteq \Sigma^{b_i}$ and sends them to M. A then expects to receive strings x_i and y_i and number b_{i+1} from M. A checks that $x_i \in H^{-1}(Z)$. If not then A immediately rejects. Then A performs round $i+1$.

Final round $g+1$:

Let $s_g = x_1 \# y_1 \# \cdots \# x_g \# y_g$. A randomly selects l linear functions $H = \{h_1, \ldots, h_l\}$, $h_i: \Sigma^l \to \Sigma^{b_{g+1}}$ and l strings $Z \subseteq \Sigma^{b_{g+1}}$. It then expects to receive a string $r \in \Sigma^l$ from M and checks that $r \in H^{-1}(Z)$. A accepts if for each $i \leqslant g$ $V(w, r, s_i) = x_{i+1}$, $V(w, r, s_g) =$ accept and $\Sigma^{b_i} \geqslant l - g \log l$.

Can Merlin Convince Arthur?

Now we show that $\Pr[V(w) \text{ accepts}] > e(n)$ iff $\Pr[A(w) \text{ accepts}] \geqslant 2/3$.

$(\rightarrow)$ *Merlin's Protocol When* $w \in W$

First some notation. For $r \in \Sigma^l$ and $s = v_1 \# v_2 \# \cdots \# v_k$ a stream of messages we say

$$(V^*P)(w, r) \text{ accepts via } s$$

if the first k messages sent by V and P agree with s and $(V^*P)(w, r)$ accepts.

Suppose $\Pr[V(w) \text{ accepts}] \geqslant 2/3$. Fix any P such that $\Pr[(V^*P)(w) \text{ accepts}] \geqslant 2/3$. We now exhibit a protocol for M such that $\Pr[(A^*M)(w) \text{ accepts}] \geqslant 2/3$.

Round 0:

Let $i = 1$. Proceed with "obtain b_1."

Obtain b_i $(i \leqslant g)$:

Let $s_{i-1} = x_1 \# y_1 \# \cdots \# x_{i-1} \# y_{i-1}$ be the message stream for the V–P protocol produced so far. For each $x \in \Sigma^m$ let $\alpha_x = \{r: (V^*P)(w, r)$ accepts via $s_{i-1} \# x\}$. Group these α's into l classes $\gamma_1, \ldots, \gamma_l$ where γ_d contains α's of size $> 2^{d-1}$ and $\leqslant 2^d$. Choose the class $\gamma_{\max}$ whose union $\bigcup \gamma_{\max} = \bigcup \{\alpha_x: \alpha_x \in \gamma_{\max}\}$ is largest. Send $b_i = 1 + \lceil \log |\gamma_{\max}| \rceil$.

Round i:

M receives $h_1, \ldots, h_l$ from A and strings $z_1, \ldots, z_l$. If there is an $x \in H^{-1}(Z)$ such that $\alpha_x \in \gamma_{\max}$, call it x_i. Then, M responds with the

pair x_i, y_i where $y_i = P(s_{i-1} \# x_i)$. Otherwise M responds with "failure." In the later analysis we refer to the set α_{x_i} as α_i. Set $i \leftarrow i+1$. Go to "obtain b_i."

Obtain b_{g+1}:

M produces the value b_{g+1} as follows: Let $s_g = s_{g-1} \# x_g \# y_g$ be the message stream that has been selected. So $\alpha_g = \{r: (V^*P)(w, r)$ accepts via $s_g\}$. Send $b_{g+1} = 1 + \lceil \log |\alpha_g| \rceil$.

Round $g+1$:

M receives $h_1, \ldots, h_l$ and strings $z_1, \ldots, z_l \in \Sigma^{b_{g+1}}$. If there is an $r \in \alpha_g \cap H^{-1}(Z)$, then M responds with r. Otherwise M responds with "failure." (Note that $r \in \alpha_g$ implies that $V(w, r, s_g) =$ accept.) End of Protocol.

We now show that $\Pr[(A^*M)(w)$ accepts$] \geqslant 2/3$. Let $\alpha_0 = \{r: (V^*P)(w, r) =$ accept$\}$. Since $\Pr[V$ accepts $w]$ is high, $|\alpha_0| \geqslant (2/3)2^l$. By the definition of M, A will accept provided M never responds "failure" and $\Sigma b_i \geqslant l - g \log l$. By the approximate lower bound lemma the probability that M responds failure at any round is $\leqslant (7/8)^l$. Hence, the probability that M ever responds failure is $\leqslant g(7/8)^l \ll 1/3$, since we assume that $l > \max[g, 50]$.

The following two claims show that $\Sigma b_i \geqslant l - g \log l$.

CLAIM 1. For each $0 \leqslant i < g$

$$|\alpha_i| \geqslant \frac{|\alpha_{i-1}|}{l 2^{b_i}}.$$

Proof. Consider round i and the sets α_x defined in "obtain b_i." By definition the α_x's partition α_{i-1} and hence $\bigcup_x \alpha_x = \alpha_{i-1}$. Hence

$$|\bigcup \gamma_{\max}| \geqslant \frac{|\alpha_{i-1}|}{l}.$$

Since all members of $\gamma_{\max}$ differ in size by at most a factor of 2 and since $\alpha_i \in \gamma_{\max}$ we have

$$|\alpha_i| \geqslant \frac{|\bigcup \gamma_{\max}|}{2|\gamma_{\max}|}$$

and since $b_i = 1 + \lceil \log |\gamma_{\max}| \rceil$ we have

$$2^{b_i} \geqslant 2|\gamma_{\max}|.$$

Thus

$$|\alpha_i| \geqslant \frac{|\bigcup \gamma_{\max}|}{2^{b_i}} \geqslant \frac{|\alpha_{i-1}|}{l2^{b_i}} \qquad \square$$

CLAIM 2. $\Sigma b_i \geqslant l - g \log l$.

Proof. By Claim 1 we have

$$|\alpha_g| \geqslant \frac{|\alpha_0|}{l^g \cdot \Pi_{i \leqslant g} 2^{b_i}}.$$

Since $|\alpha_0| \geqslant (2/3)2^l$ and taking logs

$$\log|\alpha_g| \geqslant (l-1) - \left(g \log l + \sum_{i \leqslant g} b_i\right).$$

Since $b_{g+1} \geqslant 1 + \log|\alpha_g|$

$$\sum_{i \leqslant g+1} b_i \geqslant l - g \log l.$$

(←) *Merlin's Impotence When* $w \notin W$

Show that if $\Pr[V(w) \text{ accepts}] \leqslant e$, then $\Pr[A(w) \text{ accepts}] \leqslant 1/3$.

For every $i > 0$ and $s_i = x_1 \# y_1 \# \cdots \# x_i \# y_i$ let $a(s_i) = \max_P \Pr[(V^*P)(w) \text{ accepts via } s_i]$. For each $x \in \Sigma^m$ let y_x be any $y \in \Sigma^m$ maximizing $a(s_i \# x \# y)$.

We now show that Merlin must be extremely lucky in at least one step of the protocol in order to convince Arthur to accept.

CLAIM 3. $a(s_i) = \Sigma_x a(s_i \# x \# y_x)$ $\square$

Fix $0 \leqslant i < g$ and s_i. For every $c > 0$ let $X_c = \{x : a(s_i \# x \# y_x) \geqslant a(s_i)/c\}$.

CLAIM 4. $|X_c| \leqslant c$ $\square$

Fix $b, d > 0$. Choose l random linear functions $H = \{h_1, \ldots, h_l\}$, $h_i : \Sigma^m \to \Sigma^b$ and l random strings $Z \subseteq \Sigma^b$. Pick any $x \in H^{-1}(Z)$ and any $y \in \Sigma^m$. Let $s_{i+1} = s_i \# x \# y$.

Call the following event E_{i+1}:

$$a(s_{i+1}) \geqslant \frac{a(s_i)}{2^b/d}.$$

CLAIM 5. For $i \leqslant g$ $\Pr[E_i] \leqslant l^2/d$.

Proof. Let $c = \lfloor 2^b/d \rfloor$. Then $|X_c| \leqslant 2^b/d$ by claim 4. Since $a(s_i \# x \# y_x) \geqslant a(s_{i+1})$ by the definition of y_x, if $a(s_{i+1}) \geqslant a(s_i)/(2^b/d)$ then $x \in X_c$. Since $x \in H^{-1}(Z)$, if E_{i+1} occurs then $x \in X_c \cap H^{-1}(Z)$ and so $X_c \cap H^{-1}(Z) \neq \phi$ and $H(X_c) \cap Z \neq \phi$. But

$$\Pr[H(X_c) \cap Z \neq \phi] \leqslant l^2/d$$

by the approximate lower bound lemma. □

Fix s_g. Choose l random linear functions $H = \{h_1, \ldots, h_l\}$, $h_i: \Sigma^l \to \Sigma^{b_{g+1}}$ and l random strings $Z \subseteq \Sigma^{b_{g+1}}$. Pick any $r \in H^{-1}(Z)$. Call the following event E_{g+1}:

$$2^l a(s_g) \leqslant 2^b/d \text{ and } (V^*P)(w, r) \text{ accepts via } s_g.$$

CLAIM 6.

$$\Pr[E_{g+1}] \leqslant l^2/d.$$

Proof. By the approximate lower bound lemma part 2, since $|\{r: (V^*P)(w, r) \text{ accepts via } s_g\}| = 2^l a(s_g)$. □

In any run of A and M, event E_i may occur during round i, where $b = b_i$ for $i \leqslant g + 1$. The probability that each occurs is at most l^2/d and therefore the probability that any occurs is at most $(g + 1)\, l^2/d$. Choose

$$d = 3(g + 1)\, l.$$

Then $\Pr[\exists i E_i \text{ occurs}] \leqslant 1/3$.

Assume no E_i occurs. Then we show that A will reject, provided that $\Pr[V(w) \text{ accepts}] \leqslant e$.

Since $\forall i \leqslant g, \neg E_i$, we have

$$\frac{a(s_0)}{\Pi_{i \leqslant g}(2^{b_i}/d)} \geqslant a(s_g).$$

Since $\neg E_{g+1}$:

$$(V^*P)(w, r) \neq \text{accept}$$

or

$$2^l a(s_g) \geqslant 2^{b_{g+1}}/d.$$

Thus if $(V^*P)(w, r)$ accepts, combining the above:

$$2^l a(s_0) \geqslant \prod_{i \leqslant g+1} (2^{b_i}/d)$$

so, since $l \geqslant g + 1$, taking logs:

$$\begin{aligned} l + \log a(s_0) &\geqslant \sum b_i - (g+1)\log d \\ &\geqslant \sum b_i - 4g \log l \end{aligned}$$

but

$$a(s_0) = \Pr[V(w) \text{ accepts}] \leqslant e < l^{-5g^2}$$

so

$$l - 5g^2 \log l > \sum b_i - 4g \log l.$$

Thus

$$\sum b_i < l - g \log l.$$

Recall that Arthur only accepts if $(V^*P)(w, r)$ accepts and $\Sigma b_i \geqslant l - g \log l$. Therefore if $\forall i \leqslant g + 1$, E_i does not occur and $\Pr[V(w) \text{ accepts}] \leqslant e$, then Arthur will reject. Hence $\Pr[A(w) \text{ accepts}] \leqslant 1/3$. □

5. PROBABILISTIC, NONDETERMINISTIC TURING MACHINES

We can define a new type of Turing machine that accepts precisely those languages in IP. This gives an automata theoretic characterization of this class.

DEFINITION. A *probabilistic, nondeterministic, Turing machine*, N, is defined conventionally except that it has two kinds of nondeterministic states: random states denoted  and guess states, denoted  .

Given a configuration, c, of such a machine we assign it a probability $p(c)$ of accepting as follows: if c is an accept configuration then $p(c) = 1$; if c is a reject configuration then $p(c) = 0$; if c is a deterministic configuration then $p(c) = p(c')$ where c' is the successor of c; if c is a random configuration, then $p(c)$ is the average of $p(c')$; and if c is a guess state then $p(c)$ is the maximum of $p(c')$ for c' a successor. We say that $\Pr[N(w) \text{ accepts}] = p(c_{\text{start}})$ where c_{start} is the starting configuration for N on input w. One way to think of computations on these machines is that at every random state, a coin is flipped to determine the successor and at every guess state the successor with highest probability of eventually accepting is selected.

DEFINITION. Say $W \in \text{BPNP}$ if there is a probabilistic, nondeterministic, polynomial time Turing machine N such that for all $w \in \Sigma^*$:

1. If $w \in W$ then $\Pr[N(w) \text{ accepts}] > 2/3$
2. If $w \notin W$ then $\Pr[N(w) \text{ accepts}] < 1/3$.

THEOREM. $\text{IP} = \text{AM(poly)} = \text{BPNP}$.

Proof. Immediate. These machines are just a reformulation of Arthur–Merlin games.

6. OPEN QUESTION

In [B] Babai states that $\text{AM}[2] = \{W: W \in NP^R, \text{ for almost all oracles } R\}$. However, an oversight in his argument leaves this equality an open question. (Note added in proof: This statement has recently been proved by Nisan and Wigderson [NW].)

ACKNOWLEDGMENTS

We are grateful to Oded Goldreich and Johan Hastad for pointing out that our proof works for a polynomial number of interactions. Silvio Micali's comments have been inspiring, as always. Thanks again to Oded

for a careful reading of this paper and extensive suggestions. Ron Greenberg also suggested improvements. Discussions with Laszlo Babai, Anne Condon, Jack Edmonds, and Eva Tardos were very helpful. Ed Bein, Danny Soroker, and Jeannine St. Jacques provided much appreciated last minute assistance. Research supported in part by NSF Grant 8509905 DCR (to S. G.) and in part by NSF Grant MCS-8304769 and Air Force Grant AFOSR-86-0078 (to M. S.). Much of this work took place while M. S. was at the University of California–Berkeley.

REFERENCES

[AGH] W. Aiello, S. Goldwasser, and J. Hastad, "On the power of interaction," *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, 368–379 (1986).

[B] L. Babai, "Trading group theory for randomness," *Proceedings of 17th Symposium on the Theory of Computation*, Providence, Rhode Island, 1985.

[BGGHKMR] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Hastad, J. Kilian, S. Micali, and P. Rogaway, "Everything provable is provable in zero-knowledge," to appear in the Crypto 88 conference proceedings (1988).

[BHZ] R. Boppana, J. Hastad, and S. Zachos, "Does co-NP have short interactive proofs," *Information Processing Letters* 25 (1987) pp. 127–132.

[CKS] A. Chandra, D. Kozen, and L. Stockmeye, "Alternation," *JACM* 114 (1981).

[Co] S. Cook, The Complexity of Theorem Proving Procedures, 3rd STOC, 171.

[CW] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *JCSS* 18(2) 143–154 (1979).

[F] P. Feldman, personal communication.

[FS] L. Fortnow and M. Sipser, personal communication.

[Fo] L. Fortnow, "The complexity of perfect zero knowledge," *Proceedings of the 19th Annual Symposium on Theory of Computing*, 1987.

[H] J. Hastad, personal communication.

[GMR] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proofs," *SIAM J. Comput.* v. 18, no. 1 (1989) 186–208.

[GMW] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design," *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science* (1986) 174–187.

[P] C. Papadimitriou, "Games against nature," *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science* (1983) 446–450.

[R] J. Reif, "Games with imperfect information," *JCSS* 29: 274–301 (1984).

[Si] M. Sipser, "A complexity theoretic approach to randomness," *Proceedings of the 15th Annual Symposium on the Theory of Computing* (1983) 330–335.

[St] L. Stockmeyer, "The complexity of approximate counting," *Proceedings of Symposium on the Theory of Computation*, 1984.

[ZF] S. Zachos and M. Furer, "Probabilistic quantifiers vs. distrustful adversaries," to appear.

COLLECTIVE COIN FLIPPING

Michael Ben Or and Nathan Linial

1. INTRODUCTION

Randomized algorithms play an important role in parallel and distributed processing. The use of such algorithms assumes that each processor is able to generate random bits during the computation. In some applications the algorithm requires that the *same* random bit be generated by a set of processors. This task is easy if we assume that no faults may occur. We have one of the processors flip a coin and announce the outcome. If, however, the processor assigned to flip the coin happens to be faulty this may ruin the probabilistic requirement for our randomized algorithm. Suppose, then, that each processor is equipped with a fair coin, how can they generate a global coin flip that is only slightly biased despite failure of some of the processors?

This problem was considered before, mostly in the framework of the Byzantine Generals Problem [ACGM, BE, BD, Br, BR, Ra, Ya].

Advances in Computing Research, Volume 5, pages 91–115.

ISBN: 0-89232-896-7

These past solutions are all based on the assumption that information may be communicated so that only some of the parties can read it. This is achieved either by choosing an appropriate model of communication, or by resorting to cryptography. We want to avoid such assumptions. Technically this means that we deal only with games of complete information.

The most obvious approach to solve this question is via what we call a one-round coin flipping scheme: Say that there are n processors involved. Fix a Boolean function f: $\{0,1\}^n \to \{0,1\}$. Whenever a random bit is needed, instruct each processor P_i to flip his coin and announce the outcome x_i. The global coin flip is taken as $f(x_1, \ldots, x_n)$. How sensitive is this approach to the possible existence of faults? If the processors act simultaneously, the situation is very favorable. If f is the parity function, then clearly, even if only one processor is in order, this yields a perfect coin flip. In a distributed environment, where one cannot assume perfect simultaneity, the parity function is not very useful. A single faulty processor that announces his bit already knowing the bits announced by all the rest has complete control over the global bit.

We are thus looking for Boolean functions on which every variable has only a small influence. The discussion above hints at the features we should expect from measures of influence. Indeed there are a number of inequivalent such quantities. We present the simplest and probably the most natural one among them: Let $S \subseteq \{1, \ldots, n\}$ be a subset of the variables of the Boolean function f. Randomly set the variables outside S by independent perfect coin flips. This partial setting may already determine the value of f regardless of the values of the variables in S. A measure for the influence of S on f is the probability that this does not happen and the variables in S "have the last word" in determining f. Our goal is to find functions f for which this measure of influence is as small as possible for all subsets $S \subseteq \{1, \ldots, n\}$.

The assumption that the variables in S are set after those in $\{1, \ldots, n\} \setminus S$ is obviated by the following observation (Lemma 2.1): For every Boolean function f there is a monotone function g such that

$$Pr(f = 0) = Pr(g = 0)$$

and every $S \subseteq \{1, \ldots, n\}$ influences f at least as much as it influences g. So for our purposes g is at leat as good as f. This lemma thus says that it suffices to consider monotone f's. This observation removes

the need for S to be set only after the other variables. All the information we need in this case is embodied in the probability of $f = 0$ when all variables in S are set zero (resp. one).

Some very basic questions regarding Boolean functions thus arise. They also turn out to be rather natural questions in game theory. A Boolean function is for the game theorist a *simple game*. Variables are *players* and sets of variables are *coalitions*. The measure of influence of a player (the case $|S| = 1$) is a quantity already considered in game theory, under the name of *Banzhaf–Coleman power index*. Much attention has been given in game theory to measures of influence of players and sets of players in games [Ow]. The most important among these is the Shapley value. For us the Shapley value does not seem to be the most appropriate.

Consider the Boolean function $f(x_1, \ldots, x_n) = x_k$, which in game theory is the dictatorship of player k. Here, the influence (Benzhaf index) of the dictator k is 1 and is 0 for all the other players. The influence of each variable on the majority function is $\Theta(1/\sqrt{n})$. It may be thought that this is better but this is not the case. We present a Boolean function for which the influence of each variable is only $O(\log n/n)$. On the other hand it follows from known facts in either game theory [*Ha*] or combinatorics [*H*] that for any function the average influence of a variable is at least $\Omega(1/n)$. This gap between the upper and lower bound is very intriguing and our conjecture is that the upper bound is closer to the truth. Put informally, there is always some player who affects the outcome of the game in a disproportionate manner.

Until this point we restricted our discussion to single-round schemes. The reader can certainly think of more elaborate schemes for collective coin flipping based on more complicated protocols than just an application of a Boolean function. We make the observation (Proposition 5.2) that the most general coin flipping scheme can be described as follows: There is a rooted binary tree T whose leaves are labeled by zeros and ones. Each internal node of T is labeled by the name of one of the players. We start at the root. Whenever an internal node is reached, the player at this node flips a coin and announces the outcome. According to this outcome the game proceeds to either the right or the left son of that node in the tree. When a leaf is reached the game terminates and the outcome of the game is the 0/1 label of the leaf.

Such a labeled tree is called a *multiround coin flipping scheme*. This is a more complicated setup and we need to consider at least

two quantities which measure influence. Let $S \subseteq \{1, \ldots, n\}$ be a set of players. Assume that the players not in S do follow the rules and flip perfect coins. Then there is a best strategy for S to maximize the probability of a zero outcome. The difference between this probability and the probability of zero if members of S, also, play randomly measures the influence of S toward zero. A similar quantity is defined for an outcome of one.

Our main relevant result (Theorem 5) says that in every multi-round game there is a player with an $\Omega(1/n)$ influence toward one. Of course the same holds toward zero. Unfortunately, we have no similar result to guarantee the existence of a player with a substantial influence both towards zero and one. Unlike the one-round situation $\Omega(1/n)$ bound is known to be best possible. We exhibit games showing the tightness of this bound.

More general questions regard the influence of sets of players in both single and many round games. Particularly interesting is the following notion of *ε-control*. For $\varepsilon > 0$ a coalition S has ε-control over a game G if S's influence on the outcome of G is $\geqslant \varepsilon$. (As mentioned before we deal with a number of measures for influence and the definition of ε-control clearly depends on the measure at hand.) The goal is of course to find games which are not ε-controlled by small sets.

To illustrate this notion we remark that majority is ε-controlled by $O(\varepsilon\sqrt{n})$ players. We construct single-round games in which $\Omega(\varepsilon n^{0.63\ldots})$ players are needed to achieve ε-control. Recently Saks [S] analyzed a multistage game where $\Omega(n/\log n)$ players are required to achieve ε-control. We conjecture that Saks' construction is essentially the best possible. This means that there is always a negligible minority of the players that almost determines the outcome assuming all the others play randomly. Even the one-round version of this question is open and very intriguing: Given a Boolean f: $\{0,1\}^n \to \{0,1\}$ with $Pr(f = 0) = 1/2$, is there a set of $o(n)$ variables such that even if the assignment of values to all other variables is known there is a constant probabiliity that this does not yet determine f?

In the results described here we always assume that the set of faulty processors is fixed during the game. A completely different question arises if the faulty processors are determined by an adversary during the course of the game. In the single-round context it can be shown that the most robust scheme against such an adversary is the majority voting scheme. This is an easy

consequence of the isoperimetric inequality in the cube [H]. In the multistage case this problem is left open. A special case of this question was recently settled in a paper of Lichtenstein, Linial, and Saks [LLS].

2. ONE ROUND COIN FLIPPING SCHEMES—PRELIMINARIES

As explained in the introduction a one-round coin flipping scheme is nothing but a Boolean function f: $\{0,1\}^n \to \{0,1\}$. Unless otherwise stated f is usually assumed to satisfy $|f^{-1}(0)| = |f^{-1}(1)| = 2^{n-1}$ [or equivalently $Pr(f = 0) = 1/2$]. With a slight change of notation f may be thought of as a function from the power set of $\{1,\ldots,n\}$ into $\{0,1\}$. Consequently we speak of $f(A)$ for $A \subseteq \{1,\ldots,n\}$. We now set to formally define our notions of influence for a set of variables of f.

Let

$$Q_0 = Q_0(f,S) = \{A \mid A \cap S = \varnothing \text{ and for every } B \subseteq S, f(A \cup B) = 0\}$$

$$Q_1 = Q_1(f,S) = \{A \mid A \cap S = \varnothing \text{ and for every } B \subseteq S, f(A \cup B) = 1\}.$$

Also

$$q_0 = q_0(f,S) = |Q_0(f,S)| \cdot 2^{|S|-n}$$

$$q_1 = q_1(f,S) = |Q_1(f,S)| \cdot 2^{|S|-n}$$

$$q_2 = q_2(f,S) = 1 - q_0 - q_1.$$

It is seen that $q_0(q_1)$ is the probability that the values assigned to variables outside S already set f to zero (one). The probability that this partial assignment does not determine f is q_2. Accordingly, the *influence of* S on f is defined as

$$I_f(S) := q_2(f,S).$$

The *influence of S on f towards zero* (*one*) is defined to be

$$I_f^0(S) := q_0(f,S) + q_2(f,S) - Pr(f = 0),$$

$$I_f^1(S) := q_1(f,S) + q_2(f,S) - Pr(f = 1),$$

respectively. Note that $q_0(f,S) + q_2(f,S)$ is the probability that $f = 0$ assuming the players in S try to set f to zero. The influence toward zero is defined as the excess of this probability over $Pr(f = 0)$. For $1 \leqslant r \leqslant n$ we let

$$I_f(r) := \max\{I_f(S) \mid |S| = r\}$$

and similarly for $I_f^0(r)$, $I_f^1(r)$. Also $I_f := I_f(1)$.

If $S = \{x\}$ is a singleton $I_f(\{x\})$ is a quantity that was studied in game theory. A monotone Boolean function $v: \{0,1\}^n \to \{0,1\}$ is called a *monotone simple game* (N, v) with $N = \{1,\ldots,n\}$. Variables are called *players* and sets of players are *coalitions*. In this context $I_f(\{x\})$ is called the *Banzhaf–Coleman power index* of the player x, see [Ow]. We freely interchange between these two equivalent terminologies, according to the context. In game theoretic terms one of our main problems is thus to find simple games where all Banzhaf indices are small. Our first observation about this problem is that monotone Boolean functions are as good as any other:

LEMMA 2.1. Let $f: \{0,1\}^n \to \{0,1\}$ be a Boolean function. Then there exist a monotone function $g: \{0,1\}^n \to \{0,1\}$ such that

$$Pr(f = 0) = Pr(g = 0)$$

and for every $S \subseteq \{1,\ldots,n\}$

$$I_f^0(S) \geqslant I_g^0(S),$$

$$I_f^1(S) \geqslant I_g^1(S),$$

$$I_f(S) \geqslant I_g(S),$$

Proof. The proof is based on a standard technique in the extremal theory of finite sets viz. compression. The function f is made more and more monotone until eventually g is reached. Let us pick a $1 \leqslant x \leqslant n$ and consider $\tilde{f}: \{0,1\}^n \to \{0,1\}$ which is

obtained from f as follows: Suppose that for some $A \subseteq \{1, \ldots, n\}$,

$$f(A) = 1 \qquad \text{and} \qquad f(A \cup \{x\}) = 0.$$

Then we set

$$\tilde{f}(A) = 0 \qquad \text{and} \qquad \tilde{f}(A \cup \{x\}) = 1.$$

Otherwise $\tilde{f}$ equals f. We keep doing the same thing with $\tilde{f}$, but with respect to a different $1 \leqslant y \leqslant n$. It is easily seen that as long as f is not monotone there is an x for which $\tilde{f} \neq f$. Also after a finite number of such transformations the function becomes monotone. It is also clear that $Pr(f = 0) = Pr(\tilde{f} = 0)$.

We want to show that $q_0(f, S) \leqslant q_0(\tilde{f}, S)$ for all S. Let us start with the case $x \in S$. In this case we show that

$$Q_0(f, S) \subseteq Q_0(\tilde{f}, S).$$

Suppose to the contrary that there is a set $A \in Q_0(f, S) \backslash Q_0(\tilde{f}, S)$. Since $A \notin Q_0(S, \tilde{f})$ there is a set $B \subseteq S$ with $\tilde{f}(A \cup B) = 1$. But $A \in Q_0(f, S)$ and so $f(A \cup B) = 0$. The definition of $\tilde{f}$ implies that $f(A \cup B \backslash \{x\}) = 1$ and again the fact that $A \in Q_0(f, S)$ implies that $x \in A$. But this contradicts $x \in S$.

Now we have to prove our claim for $x \notin S$. Here we show that if $A \in Q_0(f, S) \backslash Q_0(\tilde{f}, S)$ then $A \backslash \{x\} \in Q_0(\tilde{f}, S) \backslash Q_0(f, S)$. Consequently the desired inequality holds. We repeat the previous arguments and conclude that $x \in A$ and $f(A \backslash \{x\} \cup B) = 1$ when $A \backslash \{x\} \notin Q_0(f, S)$. Now suppose that $\tilde{f}(A \backslash \{x\} \cup C) = 1$. But the definition of $\tilde{f}$ implies that also $f(A \backslash \{x\} \cup C) = f(A \cup C) = 1$. This, however, contradicts $A \in Q_0(f, S)$ and completes our argument.

The proof for q_1 is identical. The conclusion for I^0, I^1 and I is then immediate. □

We prove the following easy lemma:

LEMMA 2.2. (a) Let f: $\{0, 1\}^n \to \{0, 1\}$ be a Boolean function and let $x \in \{1, \ldots, n\}$. Then

$$I_f\{x\}) = 2^{1-n} \sum_{x \notin S} |f(S \cup \{x\}) - f(S)|.$$

(b) If f is monotone, let $d(x) = |\{S \subseteq \{1, \ldots, n\} | x \in S$ and

$f(S) = 0\}|$ then

$$I_f(\{x\}) = 2\,Pr(f = 0) - 2^{-n}d(x).$$

Proof. (a) This is just a restatement of the definition. (b) Let us denote

$$C_0 = \{S | x \notin S, f(S) = 0\},\ C_1 = \{S | x \in S, f(S) = 0\}.$$

From (a) and the monotonicity of f it follows that

$$I_f(\{x\}) = (|C_0| - |C_1|)2^{-n+1}.$$

But $|C_0| + |C_1| = 2^n Pr(f = 0)$. Divide by 2^{n-1} and subtract to deduce (b). □

3. ONE ROUND GAMES—A CONSTRUCTION

This section presents a one round coin flipping scheme among n players in which the influence of any particular player is only $O(\log n/n)$.

THEOREM 3. There are one round coin flipping schemes $G = G_n$ where

$$I_G = O\left(\frac{\log n}{n}\right)$$

Proof. To describe the idea of this construction let us ignore for a while issues of integrality and divisibility. For given n let b be the (unique) solution of the equation

$$(2^b - 1)^{1/b} = 2^{1-1/n}.$$

We will later show that this b satisfies

$$b = \log n - \log\log n + O(1).$$

Now decompose $[n]$ with n/b blocks of size b and consider the ideal J of those subsets of $[\eta]$ which contain no block.

$$|J| = (2^b - 1)^{n/b} = 2^{n-1}.$$

So we consider the coin flipping scheme where $v(A) = 0$ if $A \in J$ and $v(A) = 1$ in $A \notin J$. Simply stated, the overall decision is 1 if and only if a whole block unanimously votes one and is 0 otherwise. Now let us compute the influence of an individual player. Using Lemma 2.1 we have to find $d(x)$ which is the same for every $x \in [n]$. According to the definition of J we have

$$d(x) = (2^{b-1} - 1)(2^b - 1)^{(n/b)-1}$$
$$= \frac{2^{b-1} - 1}{2^b - 1}(2^b - 1)^{n/b} = \frac{2^{b-1} - 1}{2^b - 1}|J|.$$

Hence

$$I_G = I_G(\{x\}) = [|J| - 2d(x)]2^{-n+1}$$
$$= 1 - \frac{2^b - 2}{2^b - 1} = \frac{1}{2^b - 1} = O\left(\frac{\log n}{n}\right).$$

To show the last equality go back to the relation between b, n:

$$(2^b - 1)^{1/b} = 2^{1-1/n}$$

or $n = -b/\log[1 - (1/2)^b]$. We use

$$-\ln 2\left(\frac{1}{2}\right)^b \geqslant \log\left[1 - \left(\frac{1}{2}\right)^b\right] \geqslant -\left(\frac{1}{2}\right)^{b-1}$$

when $b2^b \log e \geqslant n \geqslant b2^{b-1}$. These functions of b are increasing and therefore by evaluating them at the appropriate values of b the following bound on b results

$$|b - (\log n - \log\log n)| < 2.$$

It follows that

$$I_G = (2^b - 1)^{-1} = O\left(\frac{\log n}{n}\right).$$

To overcome the difficulties involved with b not being an integer we do as follows: We select any integer b and define α to be the real number for which

$$(2^b - 1)^\alpha = 2^{\alpha b - 1}.$$

Next we define a to be the integer nearest to α, say $a = \alpha + \varepsilon$, $|\varepsilon| \leqslant 1/2$, and set $n = ab$. The ideal J is defined as before and has size

$$|J| = (2^b - 1)^a = (2^b - 1)^{\alpha+\varepsilon} = 2^{\alpha b-1}(2^b - 1)^{\varepsilon}.$$

While $2^{n-1} = 2^{(\alpha+\varepsilon)b-1}$. So

$$\frac{|J|}{2^{n-1}} = \left(\frac{2^b - 1}{2^b}\right)^{\varepsilon} = 1 - \frac{\varepsilon}{2^b} + O\left(\frac{\varepsilon^2}{4^b}\right).$$

Therefore, by adding $2^{n-b-1}\varepsilon$ sets to J, still maintaining J being an ideal, the influence of every player can rise to at most $(1 + \varepsilon)/2^b = O(\log n/n)$. □

For completeness sake we add the following proposition:

PROPOSITION 3. Let f be a Boolean function on n variables, with $Pr(f = 0) = 1/2$. Then

$$\sum_{i=1}^{n} I_f(x_i) \geqslant 1.$$

This bound is tight as shown by $f(x_1, \ldots, x_n) = x_1$. This result is known in game theory [H]. The reader may verify it by noticing that given any set of 2^{n-1} vertices in the n-dimensional cube there are at least 2^{n-1} edges in the associated cut. Theorem 5 gives a more general result. Meanwhile we state:

CONJECTURE 3. For every Boolean function f on n variables with $Pr(f = 0) = 1/2$ there is a variable x such that

$$I_f(x) = \Omega\left(\frac{\log n}{n}\right).$$

Proposition 3 implies the existence of a variable with influence at least $1/n$. Noga Alon showed, using eigenvalue methods, the existence of a variable with an influence of at least $[2 - o(1)]/n$.

4. SYMMETRIC GAMES

The following symmetry condition is commonly imposed in the context of human voting games: If every player reverses his vote the collective decision has to change as well. If our coin flipping scheme is described by a simple game the condition is that for every coalition $S \subseteq N$

$$v(S) + v(N \setminus S) = 1.$$

It also turns out that symmetric games with small influences are useful building blocks for robust coin flipping schemes that are not necessarily symmetric (see Section 6).

We would like to consider our general problem in the context of symmetric coin flipping games:

1. Find symmetric games which minimize $I_G(r)$. Notice that the symmetry condition implies that for all $S \subseteq N$, $I_G^1(S) = I_G^0(S)$.
2. Find symmetric games for which the least number of players which ε-control the game is as large as possible.

We have the following result for the symmetric case:

THEOREM 4. (a) There exist symmetric games $G_n = G$ for which the influence function satisfies

$$I_G(r) \leqslant \frac{r}{n^\alpha} \qquad \text{for} \qquad 0 \leqslant r \leqslant n^\alpha$$

where $\alpha = \log_3 2 = 0.6309\ldots$.

(b) There exist symmetric games $G_n = G$ for which the individual influence function satisfies

$$I_G \leqslant \frac{1}{n^\beta}$$

where $\beta = \log_7(32/9) = 0.6518\ldots$.

Proof. We first introduce the notion of the *composition of games*: Let $G = ([n], v)$ and $G_i = (P_i, w_i)$ be simple games, $(i = 1, \ldots, n)$, where the sets of players P_i are mutually disjoint.

The G-composition of $\{G_i\}$ is the game

$$G = \left(\bigcup_1^n P_i, w \right)$$

where

$$w(S) = 1 \quad \text{iff} \quad v(\{i \mid w_i(X \cap P_i) = 1\}) = 1.$$

Intuitively this means that the set of players is composed of n committees where the internal voting in the ith committee is by the w_i rule, and the committees votes are combined by v.

Next, we introduce some more definitions. A hypergraph H is *intersecting* if every two edges have a nonempty intersection. H is *two-colorable* if there is a partition of the vertices $V = V_1 \cup V_2$ such that no edge is contained in either V_1 or V_2. Finally, we say that a game $G = (N, v)$ is *transitive* if there is a transitive group acting on N under which v is invariant.

PROPOSITION.

(a) Let $H = (V, E)$ be an intersecting, non-2-colorable hypergraph, then the coin flipping scheme $G = (V, v)$ given by

$$v(S) = 1 \quad \text{iff} \quad \text{there is an } A \in E \text{ such that } A \subseteq S,$$

is symmetric.

(b) If $G = ([n], v)$ is a symmetric coin flipping scheme and so are $G_1, \ldots, G_n$ then the G composition of $\{G_i\}$ is also symmetric.

(c) Let $G = ([n], v)$ and $H = H_1 = H_2 = \cdots = H_n$ be symmetric transitive games and let K be the G composition of $\{H_i\}$, then K is also transitive and

$$I_K = I_G I_H.$$

Proof. (a), (c), and the transitivity of K in (c) are simple. Since K is transitive $I_K = I_K(\{x\})$ for any player x in K. We may therefore assume that x is some player in $H_1 = H$. Now $I_K(\{x\})$ is the probability that the game is not determined by the votes of all other

players. This probability is exactly the probability that H_1 is not determined by the votes of other players in H_1, that is I_H, times the probability that K is not determined by the outcome of the games $H_2, \ldots, H_n$. Since the players in $H_2, \ldots, H_n$ flip coins to set their votes and $Pr(H = 0) = Pr(H = 1) = 1/2$ the probability that K is not determined by the "votes" of $H_2, \ldots, H_n$ is I_G, and thus

$$I_K = I_H I_G. \qquad \square$$

Consider the following two examples:

1. Let $n = 2t - 1$ and consider the hypergraph of all t-sets of $[n]$: It is clearly intersecting, non-2-colorable and the game it determines (i.e., majority voting) is transitive.
2. The hypergraph of lines in the Fano plane. It has 7 vertices and the edges are $\{(1,2,4), (2,3,5), (3,4,6), (4,5,7), (1,5,6), (2,6,7), (1,3,7)\}$. It is easy to check directly that it is interesting, non-2-colorable, and transitive. For readers familiar with projective geometry only non-2-colorability needs elaboration. This is true because any set of 4 points that does not contain a line is the complement of a line in this plane.

Using these examples we can now prove our theorem.

Part (*a*). Let H_1 be the majority game of 3 players. Define H_k recursively as the H_1 composition of three copies of H_{k-1}. Denote by $n = 3^k$ the number of players in H_k and let $J(n,r) = I^1_{H_k}(r)$ be the largest influence toward 1 that r players can have in H_k, that is, the probability of outcome 1 if all the r players vote 1 minus $Pr(H_k = 1) = 1/2$. Clearly there is a set of $2^k = n^\alpha$ players that completely control H_k so only $r \leqslant n^\alpha$ is of interest. We prove by induction on k that for such r

$$J(n,r) \leqslant \frac{r}{2n^\alpha}.$$

This is true for $k = 1$ as can easily be verified.

To proceed we consider how these r players are split among the three H_{k-1} component of H_k. Say there are r_i of them in the ith component $i = 1, 2, 3$. We are allowed to assume that for all i

$$0 \leqslant r_i \leqslant \left(\frac{n}{3}\right)^\alpha = \tfrac{1}{2}n^\alpha,$$

since $(n/3)^\alpha$ players can have complete control over H_{k-1}. The condition $\Sigma r_i = r$ must clearly hold too.

The best strategy for the r players in order to achieve an outcome of 1 in the game is for the r_i players in the ith component to play toward 1 in their component. The probability for the game to end with a 1 under such strategy is the probability that at least two of the components end with 1. The probability is, therefore,

$$\begin{aligned}
&[\tfrac{1}{2} + J(n/3, r_1)][\tfrac{1}{2} + J(n/3, r_2)][\tfrac{1}{2} + J(n/3, r_3)] \\
&+ [\tfrac{1}{2} + J(n/3, r_1)][\tfrac{1}{2} + J(n/3, r_2)][\tfrac{1}{2} - J(n/3, r_3)] \\
&+ [\tfrac{1}{2} + J(n/3, r_1)][\tfrac{1}{2} - J(n/3, r_2)][\tfrac{1}{2} - J(n/3, r_3)] \\
&+ [\tfrac{1}{2} - J(n/3, r_1)][\tfrac{1}{2} + J(n/3, r_2)][\tfrac{1}{2} - J(n/3, r_3)] \\
&= \tfrac{1}{2} + \tfrac{1}{2} \sum H(n/3, r_i) - 2 \prod J(n/3, r_i) \\
&\leqslant \tfrac{1}{2} + \tfrac{1}{2} \sum J(n/3, r_i).
\end{aligned}$$

Therefore

$$J(n, r) \leqslant \tfrac{1}{2} \max \sum J(n/3, r_i)$$

where the maximum is over all choices of r_1, r_2, r_3 with $0 \leqslant r_i \leqslant (n/3)^\alpha = \frac{1}{2}n^\alpha$, $\sum r_i = r$.

By induction

$$J(n/3, r_i) \leqslant \frac{r_i}{2(n/3)^\alpha} = \frac{r_i}{n^\alpha}$$

and so

$$J(n, r) \leqslant \frac{1}{2} \sum \frac{r_i}{n^\alpha} = \frac{r}{2n^\alpha}$$

as claimed. The conclusion about ε-control follows by solving $J(n, r) \geqslant \varepsilon$ for r.

Part (b). The construction here is similar to the construction above with the building block being the Fano game rather than the majority of three game. Let $F = F_1$ be the Fano game on seven players, and inductively define F_k to be the F composition of seven copies of F_{k-1}. Let $n = 7^k$ be the number of players in F_k. It is easy to see that $I_F = 9/32$, and thus

$$I_{F_k} = \frac{9^k}{32^k} = \frac{1}{n^\beta}. \qquad \square$$

5. MULTISTAGE GAMES—LOWER BOUNDS

We have already mentioned in Proposition 3 a limitation of single round coin flipping schemes: There is always a player with an $\Omega(1/n)$ influence on the outcome. One can certainly device more elaborate schemes for n players to flip a coin and hope to reduce the influence of any of the participants on the outcome. As it turns out, the same limitation still prevails for much more general schemes of complete information. The most general coin flipping scheme we consider here can be described as follows:

DEFINITION. (a) Let $X_1, \ldots, X_n$ be finite probability spaces, and let V be a nonempty set. A V-valued random variable $f: \prod X_i \to V$ is called a *choice function* for the players $\{1, \ldots, n\}$ on V. We say that the choice function f is *controlled* by player i if f depends only on its ith coordinate. We say that f is a *coin flip by player* i if f is controlled by player i, $V = \{v_1, v_2\}$ is of size two, and $Pr(f = v_1) = Pr(f = v_2) = 1/2$.

(b) A *general coin flipping scheme* (T, N) is a rooted tree T of finite depth with leaves labeled 0 or 1 and internal nodes labeled by choice functions (for the players N) on the set of their children. To determine the coin flip by this scheme we start at the root. Whenever an internal node is reached use the choice function at the node to select one of its children and move down the tree to that node. Continue in this manner until a leaf is reached. The label at this leaf determines the outcome of the coin flip.

(c) A *restricted coin flipping scheme* is a general coin flipping scheme (T, N) where the choice function attached to each internal node is controlled by one of the players, and a *Boolean coin flipping scheme* is a restricted coin flipping scheme where all the choice functions are just coin flips by one of the players.

We now define the notion of influence in this general context. Let (T, N) be a general coin flipping scheme and let $S \subseteq N$. Denote by $Pr(T = 0)$ the probability that the outcome of T is zero, assuming that at each internal node v the choice function f_v is used to select the next node, and that all the players pick their assignments to f_v according to the prescribed probability distributions. Now assume that at each node the players outside of S first select their assignments using the given probability distributions and then, given these assignments, the players in S can select their assignments to the choice function at the node according to their best strategy to

maximize the probability of outcome 1. The probability of S failing and T ending with 0 despite the effort by S is denoted by

$$q_0 = q_0(T, S).$$

The definition of q_1 is symmetric. The reader can verify that if T has a single internal node labeled by a single round scheme then the q_δ defined here are the same as those defined in Section 2 for the single round case. In the same way we define the influence of S in T toward zero (one) to be

$$I_T^0(S) = 1 - q_1(T, S) - Pr(T = 0),$$

$$I_T^1(S) = 1 - q_0(T, S) - Pr(T = 1).$$

Also for $1 \leqslant r \leqslant n$, and $\delta \in \{0, 1\}$,

$$I_T^\delta(S) = \max_{|S|=r} I_T^\delta(S)$$

and for $r = 1$

$$I_T^\delta = I_T^\delta(1).$$

Note that $1 - q_1(T, S)$ is the probability that the outcome is 0 when the players in S play their best strategy to reach 0. The difference between this and the probability that the outcome is 0 when all players play according to the scheme T is defined to be the influence of S toward 0.

Unlike the one round schemes that were studied in game theory and in the early days of computer science (e.g., in the context of threshold functions [Wi]), the influence of players in general coin flipping schemes, to the best of our knowledge, has not been studied before. For example, the following basic question has not been answered before: Can we approximate an unbiased coin as well as we wish despite the intervention of one of the players, by a long enough game. Or using our notation, given n and $\varepsilon > 0$, is there a general coin flipping scheme T for n players with $Pr(T = 1) = Pr(T = 0) = 1/2$ such that $I_T^0, I_T^1 < \varepsilon$.

In the following theorem we answer this question negatively, by showing that in any general coin flipping scheme for n players, for

any r, $1 \leqslant r \leqslant n$, there is always a set S of r players that can bias the coin by $\Omega(r/n)$. We wonder whether this natural result is a consequence of some more general principle. In contrast, consider the example at the beginning of Section 6. It shows an election scheme that cannot be biased by any single player.

THEOREM 5. Let (N, T) be an n player general coin flipping scheme. Let $\delta \in \{0, 1\}$ and let $p = Pr(T = \delta)$. For any r, $1 \leqslant r \leqslant n$

$$I_T^\delta(r) \geqslant \frac{r}{n} p \ln \frac{1}{p}.$$

In particular if $Pr(T = 0) = Pr(T = 1) = 1/2$, there are subsets S_0 and S_1 of N cardinality r with

$$I_T^0(S_0) \geqslant c\frac{r}{n}$$

$$I_T^1(S_1) \geqslant c\frac{r}{n}$$

where $c = (\ln 2)/2 = 0.34657\ldots$.

Proof. Our first observation is that for the purpose of lower bounds it is enough to consider only restricted schemes. This follows immediately from

PROPOSITION 5.1. For any general coin flipping scheme (T, N) there is a restricted scheme $(\tilde{T}, N)$ such that $Pr(T = 0) = Pr(\tilde{T} = 0)$ and for every $S \subseteq N$ and $\delta = 0, 1$

$$I_T^\delta(S) \geqslant I_{\tilde{T}}^\delta(S).$$

Proof. The idea is very simple: Instead of assigning values to the variables of the choice function at each node simultaneously we do this sequentially. This can only reduce the influence of any set of players. Let u be an internal node of T and let f_u: $\prod S_i \to V$ be its choice function. We replace this node by a tree T_u of $n + 1$ level as follows: All the nodes of T_u will have the given probability spaces X_i as their attached probability spaces. The root of T_u will be the

node u and the set of its children will be the set X_1. The choice function attached to this root will be the function $f_u^1(x_1, \ldots, x_n) = x_1$. In the same manner, the set of children of all nodes at the ith level of T_u will be the set X_i, with the choice function $f_u^i(x_1, \ldots, x_n) = x_i$. In other words, all the nodes of depth i are controlled by player i, and his action at this level is to select his assignment to f_u. With each leaf of T_u we can associate the n-tuple $(x_1, \ldots, x_n)$ according to the path that leads from the root u to the leaf. At this leaf we attach a copy of the node $v = f_u(x_1, \ldots, x_n) \in V$.

To construct $\tilde{T}$ we begin at the root u of T and replace it by the tree T_u. At each leaf of T_u that is labeled by $v \in V$ we put a copy of the subtree rooted at the child v of u. We now proceed with each subtree in the same manner. This way $\tilde{T}$ is constructed.

Let $S \subseteq N$ be any subset of the players. In evaluating $I_T^\delta(S)$, at each node of T we set the variables in S after the other variables have been set at random. In $\tilde{T}$ this may not be possible just because members of S may precede other players outside of S. Thus in scheme $\tilde{T}$ the players in S may have less strategies to choose from than in the scheme T. (In fact for $S = \{n - |S| + 1, \ldots, n\}$ they have exactly the same set of strategies to play.) Since any strategy for the players in $\tilde{T}$ can be used as a strategy for T it is clear that $I_{\tilde{T}}^\delta(S) \geqslant I_T^\delta(S)$. □

Our next observation is that it is enough to prove the theorem for Boolean schemes:

PROPOSITION 5.2. Let T be a general n player coin flipping scheme and let $\varepsilon > 0$. Then there exists a Boolean scheme $\tilde{T}$ such that $|Pr(T = 0) - Pr(\tilde{T} = 0)| < \varepsilon$ and for every coalition S and $\delta \in \{0, 1\}$, we have

$$I_{\tilde{T}}^\delta(S) \leqslant I_T^\delta + \varepsilon.$$

Sketch of Proof. W.l.o.g. assume T is a restricted scheme. To construct $\tilde{T}$ simply approximate the random variable at each node of T by a dyadic approximation and in a similar manner to the proof of Proposition 5.1, replace the action of the player at this node by a sequence of coin flips. □

We now return to the proof of Theorem 5. Proposition 5.2 allows us to assume that T is a Boolean scheme. Consider first the influence of one player (i.e., $r = 1$). For a node in the game tree T we consider

the game H of the subtree below it. Let $a_i = 1 - q_0(H, \{i\})$ denote the largest probability that this game ends with a 1 under i's best strategy and let $a = Pr(H = 1)$ be this probability under random play by all the players. We prove

LEMMA 5.3. For every node in a game tree

$$a_1 \cdots a_n \geqslant a^{n-1}.$$

Proof. We prove this by induction on the height in the tree. In a leaf all a_i and a are either zero or one. Let u be the father of v and w and say w.l.o.g. that u is controlled by player 1. We use b_i, c_i to denote the appropriate quantities at v, w. We have

$$b_1 \cdots b_n \geqslant b^{n-1},$$

$$c_1 \cdots c_n \geqslant c^{n-1}.$$

$$a_1 = \max(b_1, c_1) \qquad \text{say } a_1 = b_1.$$

$$a_i = \tfrac{1}{2}(b_i + c_i) \qquad n \geqslant i \geqslant 2$$

$$a = \tfrac{1}{2}(b + c).$$

We wish to show

$$a_1 \cdots a_n \geqslant a^{n-1}.$$

That is

$$b_1\left(\frac{b_2 + c_2}{2}\right)\cdots\left(\frac{b_n + c_n}{2}\right) \geqslant \left(\frac{b + c}{2}\right)^{n-1}$$

or

$$b_1(b_2 + c_2)\cdots(b_n + c_n) \geqslant (b + c)^{n-1}.$$

Expand the product $\prod_{i=2}^{n}(b_i + c_i)$ and consider the $\binom{n-1}{t}$ terms with t b -factors and $(n - t - 1)$ c-factors. This gives

$$\sum_{\substack{A \subseteq \{2..n\} \\ |A| = t}} \prod_{i \in A} b_i \prod_{j \notin A} c_j$$

By the arithmetic–geometric inequality this is greater or equal to

$$\binom{n-1}{t}\left[\prod_{\substack{A\subseteq\{2..n\}\\|A|=t}}\prod_{i\in A} b_i \prod_{j\notin A} c_j\right]^{1/\binom{n-1}{t}}$$

$$= \binom{n-1}{t}\left[\left(\prod_{i=2}^{n} b_i\right)^{\binom{n-2}{t-1}}\left(\prod_{i=2}^{n} c_i\right)^{\binom{n-2}{t}}\right]^{1/\binom{n-1}{t}}$$

$$= \binom{n-1}{t}\left[\left(\prod_{2}^{n} b_i\right)^{t/(n-1)}\left(\prod_{2}^{n} c_i\right)^{(n-t-1)/(n-1)}\right].$$

So we have

$$b_1(b_2+c_2)\cdots(b_n+c_n)$$

$$\geqslant b_1 \sum_{t=0}^{n-1}\binom{n-1}{t}\left(\prod_{2}^{n} b_i\right)^{t/(n-1)}\left(\prod_{2}^{n} c_i\right)^{(n-t-1)/(n-1)}$$

$$\geqslant \sum_{t=0}^{n-1}\binom{n-1}{t}\left(\prod_{1}^{n} b_i\right)^{t/(n-1)}\left(\prod_{1}^{n} c_i\right)^{(n-t-1)/(n-1)}$$

$$\geqslant \sum_{t=0}^{n-1}\binom{n-1}{t} b^t c^{n-t-1} = (b+c)^{n-1}.$$

Where $\prod b_i \geqslant b^{n-1}$ and $\prod c_i \geqslant c^{n-1}$ were used. □

The derivation of the theorem is easy now: We conclude from the lemma that at the root of T

$$\max a_i \geqslant a^{1-1/n} = a\left[1 + \frac{\ln(1/a)}{n} + O(n^{-2})\right],$$

or

$$I_T^1 \geqslant \frac{p}{n}\ln\frac{1}{p}.$$

This completes the proof for $r = 1$. The general case is treated in a similar way: Let $a_S = 1 - q_0(T, S)$ for all $S \subseteq N$ of size r, and

let $a = Pr(T = 0)$, then by an argument simmilar to the one presented above we have

$$\prod_S a_S \geqslant a^{\binom{n-1}{r}}$$

and thus

$$\max_S a_S \geqslant a^{1-r/n} > a\left[1 + \frac{r}{n}\ln(1/a)\right]. \qquad \square$$

REMARK. Our lower bound shows that there is always a player that can bias the coin by $O(1/n)$ toward the value 1. A similar result holds of course if we are interested in bias towards 0. We note that this lower bound is optimal as for any p_i and p between 0 and 1 satisfying $\prod p_i = p^{n-1}$ and $p_i \geqslant p$ for all i, we can construct a scheme T such that $p = Pr(T = 0)$ and $p_i = p + I_T^1(\{i\})$ as follows: Let $\alpha_i = p/p_i$. Each player i in N flips a biased coin with probability α_i of outcome 1. If all players get 1 the outcome of T is 1 and otherwise 0. Note that in this construction any player can bias the outcome toward 0, but the bias toward 1 is bounded. In the following section we give a construction where an influence of any player (toward 0 or 1) is only $O(1/n)$.

6. MULTISTAGE GAMES—UPPER BOUNDS

An *election scheme* (T, N) is defined like a coin flipping scheme, but unlike coin flipping schemes where the leaves of T are labeled by 0 or 1, in an election scheme they are marked by names of players from N. The game proceeds exactly like a coin flipping scheme and when a leaf is reached, the player whose name marks it is elected. We restrict our attention to election games where if everyone plays randomly each player is elected with equal probability. To measure the influence of a coalition S we assume that players outside S play randomly, while those in S play the best strategy to maximize the probability that a member of S is elected. The excess of this probability over $|S|/N$ is defined as the influence of S on the election scheme.

Here is an example of an election scheme $E = (T, N)$ where every player has zero influence. The root is controlled by player 1. Its $n - 1$ sons are controlled by $2, \ldots, n$, respectively, and 1 has to select between them with equality probability. The children of i's node are $n - 1$ leaves and all marks appear there but for i.

Player 1 is to be chosen with probability $1/n$ and all the rest with probability $(n-1)/n(n-2)$ each. The reader can easily check that under random play every player is elected with probability $1/n$ and a single player cannot increase his chance of being elected no matter what strategy he plays. The next theorem follows easily now.

THEOREM 6.1. There are multistage games $T = T_n$ such that $p(T = 0) = 1/2$ and the influence of any player x satisfies

$$I_T^1(\{x\}), I_T^0(\{x\}) = O\left(\frac{1}{n}\right).$$

Proof. The scheme may be described as follows: Run the election scheme E and have the elected player flip a fair coin. By the property of this election scheme a player may bias the coin only if he is elected. Since everyone is elected with probability $1/n$ the influence toward either zero or one are both $O(1/n)$. □

As for the influence of larger sets of players we have the following

THEOREM 6.2. There are n players multistage schemes $T = T_n$ such that $p(T = 0) = 1/2$ and for all $k, k < n^{\alpha - o(1)}$, where $\alpha = \log_3 2 = 0.63\ldots$, we have

$$I_T^0(k), I_T^1(k) = O\left(\frac{k}{n}\right).$$

Proof. We assume $n = 2^r$ and let $k \leqslant n^\alpha/2r$. Let G_0 be the n-player game of Theorem 4(a). Define the scheme $T = T_n$ to be the following: We play the game G_0 for $r = \log n$ times and let b_i be the ith outcome. The sequence $b = b_1, \ldots, b_r$ identifies one of the n players. This selected player now flips his coin again to set the outcome of T. Note that the only way a set of players can bias the coin is by trying to have one of them selected to flip the final coin. As the influence of the k players on the G_0 game is bounded by $k/2n^\alpha$, the probability that one of these k players will be reached is bounded by

$$k\left(\frac{1}{2} + \frac{k}{2n^\alpha}\right)^r \leqslant \frac{k}{n}\left(1 + \frac{1}{2r}\right)^r < \frac{2k}{n}.$$

Thus the probability of outcome 1 (or 0) is at most

$$\frac{2k}{n} + \frac{1}{2}\left(1 - \frac{2k}{n}\right) = \frac{1}{2} + \frac{k}{n}$$

and so the influence of any k players is bounded by $O(k/n)$. □

7. FINAL REMARKS AND OPEN QUESTIONS

We will now list some open problems raised in this paper along with our conjectured answers. We always refer to f as an n variable Boolean function with $Pr(f = 0) = 1/2$ and T is always an n player multiround scheme with $Pr(T = 0) = 1/2$. Most of these questions deal with the maximization of the influence function. They are classified according to the following criteria:

Single round/multiround scheme.

The influence of a single player/the gain of ε control.

The influence function considered. We deal with the following three quantities: I, $\min(I^0, I^1)$, and $\max(I^0, I^1)$.

1. As mentioned in Section 3 we conjecture that I_f is always $\Omega(\log n/n)$. In other words, every Boolean function has an influential variable.
2. Given n and $\varepsilon > 0$, what is the least r such that for every f there holds

$$\max[I_f^0(r), I_f^1(r)] > \varepsilon?$$

   We gave examples showing $r = \Omega(n^{0.63\ldots})$, but we are not sure of the best bound.
3. We conjecture that for every f there is a set S of $O(n/\log n)$ variables for which both

$$I_f^0(S), I_f^1(S) = \Omega(1).$$

   In other words, every Boolean function has a negligible set of variables with a significant influence. In Section 3 this is shown to be best possible.
4. We showed in Theorem 5 that for every T both I_T^0 and I_T^1 are at least $\Omega(1/n)$ and this is tight. What is the largest $\Psi = \Psi(n)$

such that in every T there is a player x for which both

$$I_T^0(x), I_T^1(x) \geqslant \Psi(n)$$

hold?

5. The claim of (3) is conjectured to hold also for multiround schemes. By the results of Saks [S] this, if true, is best possible.
6. We want to describe another notion of *adaptive influence* over a scheme. Let T be a given scheme, let k be an integer, and start with S an empty set. As we play the scheme T an adversary adds at most k players to S. The players currently in S play their best strategy towards δ and the rest play randomly. Let

$$A_k^\delta(T) := Pr(T \text{ ends with } \delta) - Pr(T = \delta).$$

For which T is this quantity minimized? Among the one round games this quantity is minimized for the majority function. This follows from an isoperimetric inequality in the cube [H]. A recent article of Lichtenstein, Linial, and Saks [LLS] shows that the majority function remains optimal even after unfolding the one round game to a tree scheme of n levels. It may be that majority is the answer also without extra assumptions.
7. The connection between election games and coin flipping games seems very interesting. Consider coin flipping schemes where we first run an election game and have the elected player flip a coin. It is true that the best coin flipping schemes have this form? Given a robust coin flipping scheme how can it be used to create a robust election scheme? The Vaziranis' recent results on sampling with random sources [VV] are relevant but do not seem to answer this question.

ACKNOWLEDGMENTS

We acknowledge useful discussions with A. Broder, N. Megiddo, A. Neyman, L. Stockmeyer, and U. Vazirani.

REFERENCES

[ACGM] B. Awerbuch, B. Chor, S. Goldwasser, and S. Micali, "Constructive and provably fair coin flip in byzantine networks," manuscrupt, 1984.

[BE] M. Ben-Or, "Fast asynchronous Byz antine agreement," *4th PODC* (1985).

[BD] A. Broder and D. Dolev, "Flipping coins in many pockets," *25th FOCS* (1984).

[Br] G. Bracha, "An $O(\log n)$ expected rounds randomized Byzantine generals algorithm," *17th STOC* (1985).

[BR] A. Broder, "A provably secure polynomial approximation scheme for the distributed lottery problem," *4th PODC* (1985).

[DS] P. Dubey and L. S. Shapley, "Mathematical properties of the Banzhaf power index," *Math. Oper. Res.* 4: 99–131 (1979).

[F] P. Frankel, "On the trace of finite sets," *J. Combinatorial Theory Ser. A* 34: 41–45 (1983).

[H] L. Harper, "Optimal numberings and isoperimetric problems on graphs," *J. Combinatorial Theory Ser. A* 385–393 (1966).

[Ha] S. Hart, "A note on the edges of the n-cube," *Discrete Math.* 14: 157–163 (1976).

[LLS] D. Lichtenstein, N. Linial, and M. Saks, "Some extremal problems arising from discrete control processes, to appear *19th STOC* (1987).

[Ow] G. Owen, *Game Theory*, 2nd ed. Academic Press, New York, 1982.

[Ra] M. O. Rabin, "Randomized Byzantine generals," *24th FOCS* 403–409 (1983).

[S] M. Saks, private communication.

[VV] U. V. Vazirani and V. V. Vazirani, "Random polynomial time is equal to slightly random polynomial time," *26th FOCS* 417–428 (1985).

[Ya] A. C. Yao, "On the succession problem for Byzantine generals," TR, to appear, 1984.

[Wi] R. O. Widner, "Chow parameters in threshold logic," *JACM* 18: 265–289 (1971).

ALMOST SORTING IN ONE ROUND

Miklós Ajtai, János Komlós, William Steiger, and Endre Szemerédi

ABSTRACT

N simultaneous comparisons are made between the elements of a set V of size n from an ordered universe. How large should N be if we want to learn the order relation for most pairs of elements of V?

Both the upper and the lower bound we get for N are proved by using probabilistic arguments (a random construction for the upper bound, and a random adversary for the lower bound).

1. INTRODUCTION

In a recent paper [AKSSz] we constructed an $O(\log \log n)$ parallel algorithm for finding the median of n elements, using the model of Valiant [V].[1] As most parallel comparison algorithms, it consists of

Advances in Computing Research, Volume 5, pages 117–125.

ISBN: 0-89232-896-7

several rounds, each one of which provides the answers to a large proportion of all pairwise comparisons.

This naturally leads to the problem of how much information one can get in one round, i.e., how many questions (simultaneous comparisons) should be asked to get a positive fraction (or almost all) of the possible $\binom{n}{2}$ answers. Thus, our work is a continuation of the research started by Häggkvist and Hell, Bollobás and Rosenfeld, and Bollobás and Thomason [HH], [BR], [BT].

Valiant's model attempts to separate the computation cost from the communication cost and overhead by suppressing the latter two. It is thus information theoretical in nature, but lower bounds in this strong model, of course, apply readily to more realistic models.

In the language of Valiant's model, the present problem is to determine the number of processors necessary to "almost sort" a list, i.e., to get most of the $\binom{n}{2}$ relations, in unit time (one round).

After formulating our model and the problem precisely, we propose an upper and a lower bound for the quantities in question. Interestingly enough, *both the upper and the lower bound will use probabilistic arguments*. (Typically only one of them do, the other is a "real" mathematical proof.) The lower bound will use natural averaging arguments, but the probabilistic construction for the upper bound uses an important feature of random graphs: the so-called expanding property. Consequently, the probabilistic construction can be replaced by known deterministic constructs for expanders, with a constant multiple loss in efficiency.

There have been quite a lot of parallel research done in this area. We have recently learned about papers by Noga Alon and Yossi Azar [AA], Béla Bollobás and Graham Brightwell [BB], and Nick Pippinger [P]. Their results are fairly similar to ours; some bounds are actually better than ours. In particular, [BB] disproves the conjecture made in our paper, although the right order of magnitude still seems to be "almost" as large as $n \log n \log \log n$ (see below).

We have also learned from [BB] that the question we investigate was apparently proposed by Rabin (see [BH]).

We will state the results and give the proofs in two separate sections.

2. PRELIMINARIES

2.1. The Model

Given a graph $G = (V, E)$, $|V| = n$, we interpret the *edges* of G as *questions* asked (simultaneously) about the order of V.

An *acyclic orientation* A of G is called the *raw answers* (A stands for adversary), A defines a relation on V in a natural manner: $v_1 <_A v_2$ if $\{v_1, v_2\} \in E$ and $v_1 \to v_2$ in A.

The *transitive closure* $P_G(A)$ of the relation defined by A is called the *full answers*. $P = P_G(A)$ is a partial order, but, for convenience of language, we will often disregard the diagonal elements (v, v).

The size of G is defined as the number of edges: $|G| = |E|$.

We define the size of a partial order P on a set V as the number of nondiagonal elements in P

$$|P| = \#\{(u, v);\ u, v \in V,\ u \neq v,\ (u, v) \in P\}.$$

Finally, the norm of the graph G is defined as follows:

$$\|G\| = \min_A |P_G(A)|$$

where A runs through all adversaries (acyclic orientations of G), i.e., $\|G\|$ is the number of (full) answers one gets in the worst case about the order of V for the (simultaneous) questions in E. Note that we disregard the problems involved in obtaining the full answers from the raw answers (taking transitive closure), and deal only with the information contained in A.

2.2. The Problem

How many questions do you have to ask to get almost all answers; i.e., given $\varepsilon > 0$, how large should $|G|$ be to have $\|G\| > (1 - \varepsilon)\binom{n}{2}$?

Note that $\|G\| \geqslant |G|$, and that $\|G\| = \binom{n}{2}$ iff $|G| = \binom{n}{2}$ (a discouraging fact), so one may think that you need $|G| = [1 - o(1)]\binom{n}{2}$ to get $\|G\| = [1 - o(1)]\binom{n}{2}$.

The results of Häggkvist, Hell, Bollobás, Rosenfeld, and Thomason show however that there are sparse graphs with $|G| < n^{1+\varepsilon}$ and yet $\|G\| = [1 - o(1)]\binom{n}{2}$ (an encouraging fact).

As a matter of fact, these sparse graphs provide most answers even if one can draw inference from A only by using bounded depth implications (a more realistic model than ours). The existence of these graphs was shown by probabilistic arguments. See the quoted papers for details.

2.3. Remark

$|P|$ (and thus $\|G\|$) is not necessarily the only natural measure of information. In many situations one uses the number of linear extensions of P, or other similar quantities. Our measure is most natural in the framework of selection algorithms. The reasons are explained in a forthcoming paper of Komlós and Remmel [KR] about the relation of $|P|$ and another measure—the spread of P.

2.4. More Notation

If $<_P$ is a partial order on V, then we define the lower and upper rank functions

$$\rho_x^- = \#\{v \in V;\ v <_P x\} \qquad \rho_x^+ = (|V| + 1) - \#\{v \in V;\ x <_P v\}.$$

(The closed interval (ρ_x^-, ρ_x^+) is the set of ranks of x in all possible linear extensions of P.)

Clearly,

$$\binom{n}{2} - |P| = \sum_{v \in V} (\rho_v^+ - \rho_v^-).$$

2.5. Expanders

Given positive numbers A, α, $A > 1$, $A\alpha < 1$, we say that the graph $G = (V, E)$, $|V| = n$, is an (A, α)-expander if for all $U \subset V$, $|U| \leqslant \alpha n$, we have $|N(U)| \geqslant A|U|$, where $N(U) = \{v \in V;\ (u, v) \in E$ for some $u \in U\}$ is the neighborhood of U.

G is a weak (A, α)-expander if for all $U \subset V$, $|U| \geqslant \alpha n$, we have $|N(U)| \geqslant A\alpha n$. [It is, of course, enough to check sets U, $|U| = \lceil \alpha n \rceil$, since $N(\cdot)$ is monotone; thus expanders are also weak expanders.]

Note that a weak (A, α)-expander, $\alpha = 1/(A+1)$, is simply a graph, in which there is at least one edge between any two vertex-sets of size αn.

3. THE RESULTS

In what follows, $c_1, c_2, \ldots$ are universal constants, and we assume that n is large enough so that all approximations are valid.

THEOREM 1. There are graphs with

$$|G| < f_\varepsilon(n) = \frac{c_1}{\varepsilon} n \log n [\log\log n + \log(1/\varepsilon)] \tag{1}$$

and yet

$$\|G\| > (1-\varepsilon)\binom{n}{2} \tag{2}$$

where $\varepsilon > 0$ is arbitrary (may depend on n). Moreover, most graphs with $f_\varepsilon(n)$ edges satisfy (2).

THEOREM 2.

$$\|G\| < 3n|G|/(\log n). \tag{3}$$

Thus,

$$|G| < \varepsilon n \log n \text{ implies } \|G\| < 3\varepsilon n^2$$

and hence the minimal $|G|$ with $\|G\| \approx \binom{n}{2}$ is somewhere between $cn(\log n)$ and $cn(\log n)\log\log n$.

We have strong reasons to believe that the upper bound describes the truth:

CONJECTURE. $\|G\| < c_2 n|G|/(\log n \log\log n)$.

Theorem 1 will follow from the more constructive.

THEOREM 1′. Let G be a weak (A, α)-expander graph, where $\alpha = 1/(A+1)$. Then,

$$\binom{n}{2} - \|G\| < c_3 n^2 (\log n)/A. \tag{4}$$

Now Theorem 1 is a consequence of Theorem 1′ and the following simple fact about random graphs [just choose $A = c_4(\log n)/\varepsilon$]:

Fact (random construction). Most graphs with n vertices and $c_5 A(\log A)n$ edges are weak (A, α)-expanders with $\alpha = 1/(A+1)$. (Actually, most graphs with $c(\alpha)n$ edges are weak (A, α)-expanders, where $c(\alpha) = 2h(\alpha)/\alpha^2 \approx 2A \ln A$, and $h(\alpha) = -[\alpha \ln \alpha + (1-\alpha) \ln(1-\alpha)]$.)

For proving the lower bound we will use the following result.

THEOREM 3. If the maximum valency in G is at most T, then

$$\| G \| < ne^T. \tag{5}$$

Theorem 3 will be proved by using a random adversay. In effect, we are going to prove the upper bound ne^T for the number of answers one gets in the *average*, when all $n!$ total orderings are equally likely.

This answers a question of Remmel: whether one can get, in the average, more than a linear amount of derived information from a graph with a linear number of edges. The answer is *no* as long as the graph is of bounded degree. A star shows that without bounded degrees the answer may be yes.

We will also make use of the following simple lemma.

LEMMA. Let $U \subset V$ be a subset of the vertices of G and G_U the restriction of G to U. Then

$$\| G \| \leqslant \| G_U \| + n|V - U|.$$

4. THE PROOFS

Proof of the Lemma. Let A be an adversary (for G_U) such that $P_{G_U}(A) = \| G_U \|$. Define A' to be the adversary (for G) that is identical to A on U, and that declares any vertex in $V - U$ to be larger than any vertex in U (and is arbitrary on $V - U$). Clearly,

$$\| G \| \leqslant |P_G(A')| \leqslant |P_{G_u}(A)| + n|V - U| = \| G_U \| + n|V - U|.$$

We show now that Theorem 2 is implied by Theorem 3 and the Lemma. We can assume that $|G| \geqslant n/2$, since isolated points are irrelevant. Then, for the set

$$U = \{v \in V;\ \deg(v) \leqslant T\} \qquad \text{we have } |V - U| < 2|G|/T$$

and thus

$$\|G\| \leqslant \|G_U\| + n|V - U| < ne^T + 2n|G|/T < 3n|G|/(\log n)$$

if we choose $T = \log n - \log\log n - c$.

The following *proof of Theorem 3* was offered by one of the referees, and is much simpler and more natural than our original proof.

We choose a random linear ordering L of the vertices of G, and consider the expected number N of pairs in the partial order P generated by the orientation of the graph induced by L. This *expected* number is clearly an upper bound on $\|G\|$.

Every pair in the partial order corresponds to one of possibly several directed paths in A. Hence, we let X_L denote the number of directed paths of length L in A and observe that

$$|P_G(A)| \leqslant \sum_{L \geqslant 2} X_L$$

and thus (writing E for expected value) that

$$N = E|P_G(A)| \leqslant \sum_{L \geqslant 2} EX_L.$$

There are at most NT^{L-1} paths of length L in G, and each path becomes a directed path in A with probability $1/L!$. Although things are clearly dependent, we still have that

$$EX_L \leqslant \frac{NT^{L-1}}{L!}.$$

The proof is concluded by observing that

$$\sum_{L \geqslant 2} \frac{NT^{L-1}}{L!} = \frac{N(e^T - 1 - T)}{T} < Ne^T.$$

Proof of Theorem 1′. We recall the following lemma from [AKSSz]:

LEMMA. Let $G = (V, E)$ be a weak (A, α)-expander graph on n vertices, where $\alpha = 1/(A + 1)$. Then, for any acyclic orientation of

G, the implied partial order P has the following property: for any $r \in (D, n)$ there is an $x \in V$ such that

$$r - D < \rho_x^- \leqslant \rho_x^+ \leqslant r \qquad \text{where } D = 20n(\log n)/A. \tag{6}$$

We show now that (6) implies $\binom{n}{2} - |P| < 4nD$, which amounts to Theorem 1′. (For a tighter bound see [KR].)

Let us write $d = \lceil D \rceil$, $N = \lfloor n/d \rfloor + 1$, and apply the lemma for the numbers $r_i = id$, $i = 1, 2, \ldots, N-1$, $r_N = n$. Thus we find vertices x_i satisfying

$$1 \leqslant \rho_{x_1}^- \leqslant \rho_{x_1}^+ \leqslant d < \rho_{x_2}^- \leqslant \rho_{x_2}^+ \leqslant 2d < \rho_{x_3}^- \leqslant \cdots.$$

Let us compare (in the partial order P) all $v \in V$ to these "markers" x_i. For any fixed $v \in V$, v is greater than some x_i and smaller than some others, but it is clear that the indices i (if any) such that v is incomparable to x_i, form an interval (transitivity).

Let us write m_v for the number of markers incomparable (in P) to v. Then, the total number of vertices $u \in V$ incomparable to v is

$$\rho_v^+ - \rho_v^- < (m_v - 2)\, d.$$

Indeed, if i_1 is the smallest and i_2 is the largest marker incomparable to v, then

$$(i_1 - 2)\, d < \rho_{x_{i_1-1}}^- \leqslant \rho_{x_{i_1-1}}^+ \leqslant \rho_v^- \leqslant \rho_v^+ \leqslant \rho_{x_{i_2+1}}^- \leqslant \rho_{x_{i_2+1}}^+ \leqslant (i_2 + 1)\, d.$$

Thus,

$$\rho_v^+ - \rho_v^- < (i_2 - i_1 + 3)\, d = (m_v + 2)\, d.$$

Therefore,

$$\binom{n}{2} - |P| = \sum_v (\rho_v^+ - \rho_v^-) < \left(2n + \sum_v m_v\right) d.$$

Now, $\Sigma_v m_v < ND < n + d$, since any marker x_i is compared to all but at most D of the n vertices. This proves the claim.

Proof of the Fact. The probability that in a randomly selected graph with n vertices and N edges there are two (not necessarily

disjoint) sets of vertices of size k with no edges between them, is less than

$$\binom{n}{k}^2 \binom{\binom{n}{2} - \binom{k}{2}}{N} \binom{\binom{n}{2}}{N}^{-1} < \binom{n}{k}^2 \left[1 - \binom{k}{2} \Big/ \binom{n}{2}\right]^N < \binom{n}{k}^2 e^{-Nk^2/n^2}.$$

Choosing $k = \alpha n$, $N = [2h(\alpha)/\alpha^2]n$, the right-hand side above goes to zero as n tends to infinity, since $\binom{n}{\alpha n} = o(e^{h(\alpha)n})$.

ACKNOWLEDGMENTS

We are grateful to both referees for many helpful remarks, and especially to one referee for the proof of Theorem 3 mentioned above. Research was supported by the NSF Grant DCR-8505053 (to J.K.).

NOTES

1. In Valiant's comparison-tree model the only cost is the number of comparisons made. (Other computational costs such as the cost of determining which comparisons to make are excluded in this model.)

REFERENCES

[AKSSz] M. Ajtai, J. Komlós, W. Steiger, and E. Szemerédi, "Deterministic selection in O(log log n) time," *Proc. 18th ACM STOC* 188–195 (1986).

[AA] N. Alon and Y. Azar, "Sorting, approximate sorting and searching in rounds," manuscript.

[BH] B. Bollobás and P. Hell, "Sorting and graphs," In *Graphs and Order* (I. Rival, ed.), 169–184, 1985.

[BB] B. Bollobás and G. Brightwell, "Graphs whose every transitive orientation contains almost every relation," *Israel J. Math.*, submitted.

[BR] B. Bollobás and M. Rosenfeld, "Sorting in one round," *Israel J. Math.* 38: 154–160 (1981).

[BT] B. Bollobás and A. Thomason, "Parallel sorting," *Discrete Applied Math.* 6: 1–11 (1983).

[HH] R. Häggkvist and P. Hell, "Parallel sorting with constant time for comparisons," *SIAM J. Computing* 1013: 465–472 (1981).

[KR] J. Komlós and J. Remmel, "The spread of a partial order," *Order* 4: 285–291 (1987).

[P] N. Pippinger, "Sorting and selecting in rounds," manuscript.

[V] L. Valiant, "Parallelism in comparison problems," *SIAM J. Computing* 4: 348–355 (1975).

CHROMATIC NUMBERS OF RANDOM HYPERGRAPHS AND ASSOCIATED GRAPHS

Eli Shamir[1]

ABSTRACT

The statistics of anticliques inside large sets of vertices of random hypergraphs has exponentially small deviation probabilities. This is derived by a martingale estimate. It implies the formula $\chi(G) = [1 + o(1)]d(n)/2\log d(n)$ for the [strong] chromatic number of hypergraphs almost surely, where $d(n)$ is the expected vertex valency. This holds under several models of randomization. For the associated graphs, it allows strong correlation between sets of edges in tuples up to size t. Weak chromatic numbers are also sharply located for random hypergraphs.

Advances in Computing Research, Volume 5, pages 127–142.

ISBN: 0–89232–896–7

1. INTRODUCTION

Random graphs theory is a confluence of graph theory and probability. One studies probability distributions of the interesting graph invariants (functions preserved under graph isomorphisms) under various models of randomization. A common model is the space $\mathcal{G}(n,p)$-graphs over n labeled vertices where the $\binom{n}{2}$ vertex pairs form a Bernoulli collection of independent 0–1 valued random variables, i.e., each pair independently occurs as an edge with probability p. But it is important to work with various models of randomization and relax the independence condition. This is one of the goals here.

The typical questions concerning a graph invariant $f(G)$, say for $G \in \mathcal{G}(n,p)$, are:

Concentration: Does $\lim_{n\to\infty} f(G)/f_{n,p} \to 1$, in some sense, for suitable parameters $f_{n,p}$?
Location: Where is $f_{n,p}$ located, how does it vary with (n,p)? [the "evolution" of random graphs].

The chromatic number $\chi(G)$, an invariant of fundamental importance in theory and practice, is the one studied here for graphs and hypergraphs. It is defined as the minimal r such that the vertex set V can be partitioned into r independent, i.e., "edge free" sets. Standard tools of moments evaluation seem too weak to derive concentration and sharp location of $\chi(G)$. Using a martingale approach, sharp concentration [SS] and very recently sharp location of $\chi(G)$ for a certain range of np [B2] were obtained.

In this article, we take a step beyond edge independence and beyond graphs. The main result, stated in terms of hypergraphs, is (cf. relevant definitions in Section 2):

THEOREM 1.1. *Consider the random-hypergraphs spaces* $\mathcal{H}[n,p(n),t]$ *and* $\mathcal{UD}[n,b(n),t]$. *Let* $d(n)$ *be the expected vertex valency of a hypergraph* H (*and of the associated multigraph* G_H). *It is expressed in terms of the other parameters. If* $d(n) \geqslant n^{\alpha}$, $\alpha > 6/7$, *then with probability* $1 - o(1)$ *the chromatic number* $\chi(H)$ $[=\chi(G_H)]$ *satisfies*

$$\chi(H) = [1 + o(1)]\chi[d(n)], \tag{1.1}$$

where

$$\chi[d(n)] = d(n)/2 \log d(n). \tag{1.2}$$

Each cell of size $(t+1)$ in H contributes a $(t+1)$-clique to G_H. Thus, whether edge occurrences are independent ($t = 1$) or highly correlated within tuples up to size $(t+1)$, the chromatic number of most graphs is sharply determined by the same formula (1.1)–(1.2), involving the expected valency *only*, or equivalently the total number of edges.

This robustness of $\chi(G)$, of anticlique statistics and possibly other graph invariants, with respect to relaxations in edge independence, is quite significant in practice, for getting realistic estimates of these invariants in graphs that come up in applications. In a previous article [ScS] we established robustness of the "double-jump," the location where the giant component of a graph emerges, at average vertex valency $d(n) = 1$.

The coloring process establishing Theorem 1.1 is quite simple. It keeps peeling off anticliques of about the same size r from the remaining set of vertices M, until M is quite small ($|M| < m_f$) and colorable by a few colors. The main idea is that, with probability $1 - o(1)$, the process is never blocked, since inside any $M \subseteq V(|M| \geqslant m_f)$ there is an anticlique of size r. Indeed there are many. We can choose r so that their expected number in M is $n^{1+\varepsilon}$ and sizable deviation from it is exponentially improbable.

This deviation estimate is most conveniently derived from a martingale tail estimate. This is the reason we chose $\mathscr{U}\mathscr{D}[n, b(n), t]$ as the basic model. In it, hypergraphs are compactly presented by n adjacency lists, of size $b(n)$ each, and the random generation process of H involves $n \cdot b(n)$ selections.

2. RANDOM HYPERGRAPHS MODELS

A hypergraph H over a set V of n (labeled) vertices is a family E of subsets of V. We call $e \in E$ a cell (see Remark 2.1). The size $|e| = t(e) + 1$. If $t(e)$ is constant for all $e \in E$, then it is $(t+1)$-uniform. Standard definitions are found in [Be, B1].

We find it convenient to work also with pointed cells. A vertex $x \in e$ is singled out as pointed. If $e' = e - \{x\}$ then with a slight abuse of notation $e = \{x, e'\}$ and e' is a *neighbor* of x. A pointed H is presented by the adjacency *list of neighbors*, for each $x \in V$.

Upon forgetting the pointing, an underlying hypergraph $\mathcal{U}H$ is obtained.

A subset $R \subset V$ is an *anticlique* if $|e \cap R| \leqslant 1$ for each $e \in E$. This is the strong extension of the independence notion in graphs. Weaker forms are obtained if 1 is replaced by $\gamma(e) \leqslant |e| - 1$ (cf. Section 5). Notice that vertices outside R do effect the anticlique condition for R. A (strong) coloring is a partition of V into disjoint anticliques. The chromatic number $\chi(H)$ is the minimum cardinality of such a partition.

We associate to H an ordinary multigraph G_H with the same set of vertices.

$$G_H = (V, E_G) \qquad \text{where } \{x, y\} \in E_G \Leftrightarrow \{x, y\} \subseteq e \text{ for some } e \in E. \tag{2.1}$$

Clearly, a coloring [anticlique] of H is precisely a coloring [independent set] of G_H.

REMARK 2.1. (2.1) means that a cell e contributes a clique of edges to E_G. One can envisage cells contributing (prescribed) subsets of their pairs. Our main result will hold.

The space $\mathcal{H}_{[P]}[n, p(n), t]$ of random [pointed] hypergraphs is defined as in the case of graphs ($t = 1$). Each $(t + 1)$ [pointed]-tuple e occurs with probability $p(n)$, distinct occurrence events are independent. In case of nonuniform cells, t and $p(n)$ are vectors of values. One can let the range of t grow slowly with n. For simplicity's sake, we limit ourselves to bounded t and uniform hypergraphs. Otherwise the notation becomes cumbersome but the results are the same; cf. Remark 3.2.

The total number of edges in the associated multigraph G_H is $\binom{t}{2}\binom{n}{t+1}p(n)$, the expected vertex valency is

$$d(n) = t\binom{n-1}{t}p(n); \tag{2.2}$$

this is the most convenient space-parameter for comparing various models and expressing the location of chromatic numbers and other invariants.

The relation between the models of pointed and unpointed hypergraphs is simple.

$$\mathscr{H}(n, p_1, t) = \mathscr{U}\mathscr{H}_P(n, p, t), \qquad 1 - p_1 = (1-p)^{t+1}, \quad (2.3)$$

$$\text{Prob}(H) = \sum_{\mathscr{U}H' = H} \text{Prob}(H'), \quad (2.4)$$

$\mathscr{U}$ is the pointing forgetful functor.

The pointed space can be partitioned as follows:

$$\mathscr{H}_P(n, p, t) = \bigcup_{b(v)} \left\{ \mathscr{D}[n, b(v), t] \,|\, b(v) \in B\left[p, \binom{n-1}{t}\right] \right\} \quad (2.5)$$

$$\mathscr{D}[n, b(v), t] = \{H \,|\, \text{each } v \text{ selects } b(v) \text{ random neighbors}\}. \quad (2.6)$$

The union in (2.5) is parametrized by the sample vectors $b(v)$, $v \in V$, whereby each v independently picks up its pointed valency from the binomial distribution $B\left[p, \binom{n-1}{t}\right]$. Each sample vector is included in the union with its outcome probability. In (2.6), the vector $b(v)$ is given [if it is uniform, we write $b(n)$]. Each $v \in V$ *selects* $b(v)$ *random* neighbors among the $\binom{n-1}{t}$ possible ones; this defines the (uniform) probability measure in $\mathscr{D}$.

If $b(n)$, the expected number of neighbor cells per vertex, satisfies the inequality

$$b(n) = \binom{n-1}{t} p(n) \geqslant n^{\beta}, \qquad \beta > 0$$

then all the functions $b(v)$ picked in (2.5) are sharply concentrated:

$$\text{with probability } 1 - o(1), \qquad \text{all } b(v) = [1 + o(1)]b(n). \quad (2.7)$$

This follows from standard tail estimates for the binomial distribution.

The order we follow to compute the location of $\chi(H)$ (it can be used for other purposes) is to derive it first for $\mathscr{U}\mathscr{D}[n, b(n), t]$, the hypergraph space underlying (2.6) [but all $b(v)$ are set to $b(n)$, the location formula being continuous in $b(v)$]. Then (2.3), (2.5) give it for $\mathscr{H}[n, p_1(n), t]$.

REMARK 2.2. The main advantage of the model $\mathcal{UD}$ is clear. Representation by adjacency lists instead of adjacency matrix is more compact (even more so for $t > 1$). It involves $n \cdot b(n)$ random selections.

3. ANTICLIQUES PROBABILITIES

Let $H = (V, E_H)$ be a $(t+1)$-uniform hypergraph. For $R \subseteq V$, $|R| = r$ the anticlique condition is

$$AC(R) = \{\text{for all } e \in E_H, |e \cap R| \leqslant 1\}.$$

In $\mathcal{H}(n, p_1, t)$ it is easy to see that

$$\text{Prob}\{AC(R)\} = \exp\left[-p' \sum_{k=2}^{t+1} \binom{r}{k}\binom{n-r}{t+1-k}\right], \tag{3.1}$$

$$1 - p_1 = \exp(-p'), \qquad p' = -\log(1-p_1) = p_1[1 + O(p_1)]. \tag{3.2}$$

Set

$$\sum_{k=2}^{t+1} \binom{r}{k}\binom{n-r}{t+1-k} = (1-\delta)\binom{r}{2}\binom{n-2}{t-1} \tag{3.3}$$

where $\delta = O(tr/n)$, so

$$\text{Prob}\{AC(R)\} = \exp\left[-\binom{r}{2}p^*\right], \qquad p^* = (1-\delta)p'\binom{n-2}{t-1}, \tag{3.4}$$

indeed the left-hand side of (3.3) counts tuples f with $|f \cap R| = k > 2$ once, while $\binom{r}{2}\binom{n-2}{t-1}$ counts such tuples $\binom{k}{2}$ times, but these tuples are negligible compared to the tuples with $|f \cap R| = 2$.

REMARK 3.1. $\text{Prob}\{AC(R)\}$ is computed as if the pairs (potential edges in G_H) of R are *absent* independently with probability $\exp(-p^*)(\approx 1 - p^*)$ and $p^* = (1-\delta)p'\binom{n-2}{t-1}$ sums up over all cells containing a given pair. p' is there [cf. (3.2)] since for anticliques nonoccurrence is basic.

REMARK 3.2. Similar reductions of Prob$\{AC(R)\}$ to $\mathcal{G}(n,p^*)$ with suitable p^* hold for more general cells—different sizes or contributing only fraction of their pairs.

Similar situation prevails in $\mathcal{UD}[n,b(n),t]$ as we proceed to show. Here, Prob(Event) is estimated by counting for each vertex the neighbor configurations that are in the Event. For $b(n)$ small (which is always the case if $t > 1$), it does not matter if the random selection of $b(n)$ neighbors is done with or without replacement.

For Prob$\{AC(R)\}$, the product over all vertices factors into two parts:

(i) for $v \notin R$: The selected cells should not contain $k \geqslant 2$ vertices of R;

(ii) for $v \in R$; The selected cells should not contain $k' \geqslant 1$ vertices of R.

The count of the "good" configurations in (i) is

$$\binom{n-1}{t} - \sum_{k=2}^{t} \binom{r}{k}\binom{n-1-r}{t-k} = \binom{n-1}{t} - \binom{r}{2}\binom{n-1}{t-2}(1-\delta)$$

where $\delta = O(rt/n)$. Dividing by $\binom{n-1}{t}$ we get

$$1 - \binom{r}{2}\frac{t(t-1)}{n^2}(1-\delta),$$

raising to power $(n-r)\,b(n) = nb(n)(1-\delta)$:

$$\exp\left[-\binom{r}{2}t(t-1)\frac{b(n)}{n}(1-\delta)\right].$$

Similarly for part (ii) we get

$$\exp\left[-\binom{r}{2}t\frac{b(n)}{n}(1-\delta)\right].$$

Multiplying the two, we get

$$\text{Prob}\{AC(R)\} = \exp\left[-\binom{r}{2}t(t+1)\frac{b(n)}{n}(1-\delta)\right] \tag{3.5}$$

$$= \exp\left[-r^2\frac{d(n)}{n}(1-\delta)\right]$$

$$d(n) = t[(t+1)b(n)] \tag{3.6}$$

REMARK 3.3. In $\mathscr{U}\mathscr{D}[n, b(n), t]$, $(t+1)\,b(n)$ is the expected number of cells containing a vertex v, and v has t vertex-neighbors in each cell so $d(n)$ in (3.6) is the expected vertex–vertex valency, i.e., the expected vertex valency is the associated multigraph G_H. Not surprisingly, $p^* = (1-\delta)\,d(n)/n$ serves as a substitute edge probability (more precisely e^{-p^*} is the independent nonedge probability) for G_H and Remark 3.2 also applies.

For later purposes, we need to compute probability of

$$AC(R) \text{ and } AC(S), \qquad |R| = |S| = r, \qquad |R \cap S| = l. \tag{3.7}$$

It is crucial that the conjunction probability is also computed (approximately) as if the multigraph G_H is taken from $\mathscr{G}(n, p^*)$:

$$\exp\left\{-p^* 2\binom{r}{2} + p^*\binom{l}{2}\right\}, \tag{3.8}$$

$$p^* = (1-\delta)\,d(n)/n \qquad \text{or} \qquad p^* = (1-\delta)\,p'\binom{n-2}{t-1} \tag{3.9}$$

depending on the model we work with. [Notice: the two occurrences of p^* in (3.8) have different δ in them.] Indeed, the probability of the event (3.7) is obtained by summing, for each pair $\{x, y\} \subset R$ or S the "noncell" contribution of tuples containing $\{x, y\}$ (cf. Remarks 3.1 and 3.3). But a tuple e containing pairs of R and S causes duplication. If $\{x, y\} \subset R \cap S$, the duplication is taken care of by the $p^*\binom{l}{2}$ term. Else $|e \cap (R \cup S)| \geqslant 3$. This duplication, as we have seen in (3.3) and (3.5), is taken care of by the factor $(1-\delta)$ in the $p^* 2\binom{r}{2}$ term.

4. THE NUMBER OF ANTICLIQUES

Let us fix a set of M vertices and consider $Y(M, n, r)$, the *number* of anticliques of size r in M. For EY, we multiply $\text{Prob}\{AC(R)\}$ by $\binom{m}{r}$. We take m such that

$$\log m = \log n(1-\delta'), \qquad \delta' = o(1), \qquad [\text{e.g., } m = n/\log^c n]. \tag{4.1}$$

Using (3.5),

$$EY(M,n,r) = \exp\left\{-r\left[\frac{r}{2}\frac{d(n)}{n}(1+\delta) - \log n(1-\delta') + \log r\right]\right\} \tag{4.2}$$

we substitute

$$r = (1-\varepsilon)\frac{2n}{d(n)}\log d(n), \qquad \left[\delta = O\left(\frac{tr}{n}\right) = o(1)\right], \tag{4.3}$$

then since $\log r = \log n[1 - o(1)] - \log d(n)$, the braces in (4.2) become $r(\log n)(\varepsilon - \delta'')$, $\delta'' = o(1)$ includes δ and δ' (δ' dominates!). Clearly we can accommodate ε to α to get

$$EY(M,n,r) = n^{\alpha}, \qquad -2 < \alpha < 2, \qquad r = [1 - o(1)]\frac{2n}{d(n)}\log d(n), \tag{4.4}$$

since it suffices to vary ε in $\delta'' \pm 2/r$ and by (4.3) the approximate size of r is known.

In (4.4) a threshold for anticlique size is located. Above it, they are very rare. Below it, their expected number EY is high. We proceed to show strong concentration of Y around EY. To this end, we pass to the variable $Z(M,n,r)$ that is a lower bound for Y since it counts only anticliques that do not share a pair with any other anticlique.

$$Z(M,n,r) = \sum_{R} 1 \left| \begin{array}{l} |R| = r, \quad R \subset M, \quad AC(R) \quad \textit{and if} \\ |S| = r, \quad S \subset M, \quad AC(S) \quad \textit{then } |S \cap R| \leqslant 1. \end{array} \right. \tag{4.5}$$

Given $R \subset M$ with $AC(R)$, the S-conditioning in (4.5) can be expressed by

$$X = \sum_{l=2}^{r-1} X_l = 0, \tag{4.6}$$

where

$$X_l = \text{the number of } \{S \subset M, \quad |S| = r, \quad |S \cap R| = l, \quad AC(S)\} \tag{4.7}$$

so

$$EZ(M,n,r) = \binom{m}{r} \text{Prob}\{AC(R)\} \cdot \text{Prob}\{X = 0 \mid AC(R)\} \tag{4.8}$$

$$\text{Prob}\{X = 0 \mid AC(R)\} \geqslant 1 - E[X \mid AC(R)]. \tag{4.9}$$

$$E[X_l \mid AC(R)] = \sum_{S \cap R = l} \text{Prob}\{AC(R) \textit{ and } AC(S)\}/\text{Prob}\{AC(R)\}. \tag{4.10}$$

Now we use (3.8) but upon dividing one should not forget that our equalities are approximate, up to factors $(1 - \delta)$,

$$E[X_l \mid AC(R)] = \binom{r}{l}\binom{m-r}{r-l} \exp\left\{-p^*\left[\binom{r}{2} - \binom{l}{2}\right]\right\} \cdot \exp(-\delta p^* r^2) = f_l \tag{4.11}$$

$$\delta p^* r^2 = O[d(n) r^3/n^2] = O(n^{-\varepsilon}), \qquad \varepsilon > 0 \tag{4.12}$$

by (4.14) below, so the approximation factors are very close to 1. Our goal is to show, as in [B2], that for f_l defined in (4.11),

$$\sum_{l=2}^{r-1} f_l = O(f_2 + f_{r-1}) = o(1). \tag{4.13}$$

We note the relations, based on the value of r

$$\exp(-p^* r) = d(n)^{-2}, \qquad \frac{r d(n)}{n} = 2 \log d(n).$$

Thus

$$f_2 \leqslant \binom{r}{2}\binom{m-r}{r-2} \exp\left[-p^*\binom{r}{2}\right] \leqslant \frac{r^4}{m^2}\left\{\binom{m}{r} \exp\left[-p^*\binom{r}{2}\right]\right\} = r^4 n^{\alpha} m^{-2},$$

since the expression in braces is $EY(M,n,r)$, which was taken to be n^{α}.

$$f_{r-1} \leqslant rnd(n)^{-2} = \frac{2n^2}{d(n)^3}\log d(n).$$

f_2 is more critical, because α should be kept close to 2. The restriction we impose is

$$\gamma < 1/7, \qquad \alpha = 2 - 4\gamma, \qquad d(n) = n^{1-\gamma} \qquad (\text{so } r \leqslant n^{1/7}). \quad (4.14)$$

Then clearly f_2 and f_{r-1} are $o(1)$. To estimate $\Sigma_2^{r-1} f_l$ we compute

$$f_{l+1}/f_l = \frac{(r-l)^2}{(l+1)(n-2r+l+1)}\exp[p^*(l-\delta r^2)]. \quad (4.15)$$

Now there is a $k = \theta r$ such that

(i) the ratio (4.15) rises steeply in $k \leqslant l \leqslant r-k$ (where the exponential dominates), from values well below 1 to well above 1;
(ii) in $2 \leqslant l \leqslant k$, (4.15) is $\leqslant 1/2$, in $r-k \leqslant l \leqslant r-1$ (4.15) is $\geqslant 2$.

This proves that indeed, $f_2 + f_{r-1}$ dominates Σf_l, and provided (4.14) holds, we established (4.13) and going backward to (4.8)

$$EZ(M,n,r) = EY(M,n,r)[1 - o(1)]. \quad (4.16)$$

THEOREM 4.1. *Assume* (4.14) *for* $d(n)$, (4.1) *for* m. *Then with probablity* $1 - o(1)$, *every subset* M *of size* m *contains many anticliques of size* r,

$$r = [1 - o(1)]n/\chi[d(n)]. \quad (4.17)$$

Proof. By (4.4), (4.16), there is an r of the required size such that $EY(M,n,r)$ and $EZ(M,n,r) \geqslant n^{\alpha}$. Next we show that

$$\text{Prob}\{Z(M,n,r) < \tfrac{1}{2}n^{\alpha}\} \leqslant \exp[-O(n^{1+\varepsilon})]. \quad (4.18)$$

This clearly proves the Theorem with "many" $= n^{\alpha}/2$. Martingale estimate is used to get (4.18). A hypergraph in the space $\mathcal{UD}[\mathbf{n}, b(n), p]$ is determined by selecting, in some order, $n \cdot b(n)$ neighbors. Consider the refining sequence of equivalence relations on the space

$$Q_0 < Q_1 < Q_2 < \cdots < Q_{nb(n)},$$

where H, H' are Q_j-equivalent if they made the same selections up to the jth selection. The sequence

$$Z_j = E(Z \mid Q_j), \qquad 1 \leqslant j \leqslant nb(n) \tag{4.19}$$

is a martingale and the oscillation of $|Z_j - Z_{j-1}|$ is at most $\binom{t+1}{2}$, because changing only the jth selection may effect at most $\binom{t+1}{2}$ of the pair-disjoint anticliques counted in Z [cf. (4.5)]. The length of the martingale is $nb(n) = n^{2-\gamma}$. We apply the deviation estimate of ([SS], Theorem 3) with $\lambda = \frac{1}{2}n^{\zeta}$, $\zeta = 2 - 4\gamma - 1 - \gamma/2 > 1/2$ if $\gamma < 1/7$ [cf. (4.14)], this yields (4.18).

To render the article self-contained, we outline in Section 7 a proof of the martingale deviation estimate.

5. THE CHROMATIC NUMBERS

The proof of Theorem 1.1, giving the location of the chromatic number, was indicated in the introduction. Consider the coloring process applied to H

$M := V$;
repeat
find an anticlique of size r in M, $r = [1 - o(1)]n/\chi[d(n)]$, and give it a new color
until $|M| \leqslant n/\log^2 n$;
color [the residual] M by $2d(n)/\log^2 n$ colors.

The process uses $[1 + o(1)] \cdot \chi[d(n)]$ colors, and it is almost surely not blocked, the repeat loop by Theorem 4.1, the residual coloring by Theorem 6.1, proved in Section 6.

5.1. Other Chromatic Numbers

The anticlique condition for $R \subset V$ in a hypergraph H can be weakened. Let $|\gamma(e)| \leqslant |e| - 1$:

$$IN_\gamma(R) = \{\text{for all } e \in E, \quad |R \cap e| \leqslant \gamma(e)\}. \tag{5.1}$$

Accordingly, γ-coloring [γ-chromatic number] are defined as [minimum] partition of V into IN_γ-sets. The probability of $IN_\gamma(R)$ is affected predominantly by potential $(t+1)$-cells that intersect R in $\gamma + 1$ vertices. It is simple to show that in the space $\mathscr{H}(n, p_1, t)$ the parameter replacing $d(n)$ [which served us for $\gamma = 1$, cf. (2.2)] is

$$d_\gamma(n) = \gamma \binom{t}{\gamma}\binom{n-1}{t} p(n). \tag{5.2}$$

Calculating the threshold where EY_γ crosses 1, the size of the maximal IN_γ set is found to be

$$[1 - o(1)]n\left[\frac{(\gamma+1)\log d_\gamma(n)}{d_\gamma(n)}\right]^{1/\gamma}.$$

The γ-chromatic number is, almost surely,

$$[1 + o(1)]\left\{\frac{d_\gamma(n)}{(\gamma+1)\log d_\gamma(n)}\right\}^{1/\gamma}. \tag{5.3}$$

These results are proved in the same way as for $\gamma = 1$ (strong coloring), in a sense, they are simpler and the range of density where they hold is larger. It was proved in [SSU] that the almost surely a greedy algorithm yields weak coloring ($\gamma = |e| - 1$) with number of colors expressed similar to (5.3) but without $(\gamma+1)^{-1}$ inside the braces. In [Sc], a fast algorithm for strong coloring of hypergraph is given that almost surely uses at most 4χ colors. Our process of coloring proving Theorem 1 is not fast because finding anticliques of size r is not (known to be) fast.

6. RESIDUAL COLORING

Let $M \subseteq V, |M| = m = nw(n), w(n) = o(1)$. Consider the random variable

$$F = F_M = \text{the number of edges in } G_H|M, \tag{6.1}$$

$G_H|M$ is the multigraph G_H induces on M. In our model $\mathcal{UD}[N, b(n), t]$ each V selects $b(n)$ neighbors e'.

$$\text{If } v \notin M, \qquad |e' \cap M| = s + 1 \geqslant 2,$$

$$\text{the selection gives } \binom{s+1}{2} \text{ edges,} \tag{6.2}$$

$$\text{If } v \in M, \qquad |e' \cap M| = s \geqslant 1,$$

$$\text{the selection gives } \binom{s+1}{2} \text{ edges.} \tag{6.3}$$

Since M is relatively small, only $s = 1$ really counts. The contribution from $s > 1$ can be accounted for by a factor $1 + \delta$, $\delta = O[w(n)]$. F is the sum of $(n - m)\,b(n)$ type (6.2) selections plus $mb(n)$ type (6.3) selections, F is distributed $B[\bar{\lambda}, nb(n)]$ with average probability

$$\begin{aligned}\bar{\lambda} &= \frac{(1+\delta)}{n}\binom{m}{2}\left[\frac{n^{t-1}}{(t-2)!} + \frac{2n^{t-1}}{(t-1)!}\right]\Big/\binom{n}{t} \\ &= (1+\delta)\binom{m}{2} t(t+1)/n^2.\end{aligned}$$

Thus $EF = (1 + \delta)\binom{m}{2} d(n)/n$ and standard tail estimates for the binomial-like distributions give

$$\text{Prob}\{|F - EF| \geqslant \beta(EF)\} \leqslant \exp\left[\frac{\beta^2}{2} EF\right]. \tag{6.4}$$

THEOREM 6.1. *Consider the models $\mathcal{H}$ or $\mathcal{UD}$ with expected vertex-valency $d(n)$. Let $\varepsilon > 0$. Almost surely $H_G|M$ where $|M| = m$ and $m^2 d(n) \geqslant n^2$, has average valency*

$$2F_M/m \leqslant (1 + \varepsilon)\, md(n)/n - 2. \tag{6.5}$$

If also $m^2 d^3(n) \geqslant n^4$ then almost surely each of these graphs is L-colorable with $L = (1 + \varepsilon)\, md(n)/n$ colors.

Proof. (6.5) follows from (6.4), as the condition on m and $d(n)$ make EF_M grow like n so the tail probability in (6.4) $\leqslant \exp(-kn)$.

Now if $G_H | M$ were not L-colorable, and g' the minimal graph in it which is not L-colorable, then G' a size m' L-regular graph, so its average valency is at least $L \geqslant (1+\varepsilon)\, m'd(n)/n$; this contradicts (6.5), after we check that m' is not too small. Indeed

$$m'^2 d(n) \geqslant m^2 \frac{d(n)^2}{n^2} d(n) \geqslant n^2.$$

For the proof of Theorem 1 we needed $m = n/\log^2 n$, which satisfies the requirements.

7. CONCENTRATION VIA MARTINGALES

The martingale method allows one to prove that a graph function is tightly concentrated in distribution though it does not say where this concentration occurs. A martingale is a stochastic process $x_0, \ldots, X_n$ for which $E[X_{i+1} | X_i] = X_i$. The following bound on deviations of martingales, used in the proof of Theorem 2, is recognized as Azuma inequality [A].

THEOREM 7.1. *Let* $0 = X_0, X_1, \ldots, X_n$ *be a martingale with* $|X_{i+1} - X_i| \leqslant 1$. *Then*

$$Pr[X_n > \lambda] < e^{-\lambda^2/2n}. \tag{7.1}$$

Proof. Set $Y_i = X_i - X_{i-1}$. If Y is any distribution with $E(Y) = 0$ and $|Y| \leqslant 1$ the concavity of $f(y) = e^{\alpha y}$ implies

$$E[e^{\alpha Y}] \leqslant \cosh(\alpha) \leqslant e^{\alpha^2/2}.$$

In particular

$$E[e^{\alpha Y_i}[Y_1, \ldots, Y_{i-1}] < e^{\alpha^2/2}.$$

Hence

$$E[e^{\alpha X_n}] = E\left[\prod_{i=1}^{n} e^{\alpha Y_i}\right] < e^{\alpha^2 n/2}$$

and with $\alpha = \lambda/n$

$$Pr[X_n > \lambda] = E[e^{\alpha X_n}]e^{-\alpha\lambda} < e^{\alpha^2 n/2 - \alpha\lambda} = e^{-\lambda^2/2n}.$$

The estimate (7.1) holds for $X_n - X_0$ (and for $X_0 - X_n$) in case $X_0 \neq 0$.

ACKNOWLEDGMENTS

Thanks to Jeanette Schmidt-Pruzan for most useful remarks and improvements, to Joel Spencer for inspiring conversations, and to the Computer Science department of the University of Chicago, where this work was written in 1987, for inspiring intellectual and physical atmosphere.

NOTES

1. Institute of Mathematics and Computer Science, the Hebrew University, Jerusalem.

REFERENCES

[A] K. Azuma, "Weighted sums of certain dependent random variables," *Tôhoku Math. J.* 3: 357–367 (1967).

[Be] C. Berge, *Graphs and Hypergraphs*, North Holland, Amsterdam, 1973.

[B1] B. Bollobás, *Random Graphs*. Academic Press, London, 1985.

[B2] B. Bollobás, "The chromatic number of random graphs," *Combinatorica* 8: 49–56 (1988).

[ES] P. Erdös and J. Spencer, *Probability Methods in Combinatorics*. Academic Press, New York, 1974.

[M] D. W. Matula, "Expose-and-merge exploration and the chromatic number of a random graph," to appear.

[Sc] J. P. Schmidt, "Probabilistic analysis of strong hypergraph coloring algorithm and the strong chromatic number," *Disc. Math.* 66: 259–277 (1987).

[SSU] J. Schmidt-Pruzan, E. Shamir, and E. Upfal, "Random hypergraph coloring algorithms and the weak chromatic number," *J. Graph Theory* 8: 347–362 (1985).

[ScS] J. Schmidt-Pruzan and E. Shamir, "Component structure in the evolution of random hypergraph," *Combinatorica* 5: 81–95 (1985).

[SS] E. Shamir and J. Spencer, "Sharp concentration of the chromatic number on random graphs $G_{n,p}$," *Combinatorica* 7: 121–129 (1987).

ALMOST OPTIMAL LOWER BOUNDS FOR SMALL DEPTH CIRCUITS

Johan Hastad

ABSTRACT

We give improved lower bounds for the size of small depth circuits computing several functions. In particular we prove almost optimal lower bounds for the size of parity circuits. Further we show that there are functions computable in polynomial size and depth k but that require exponential size when the depth is restricted to $k - 1$. Our Main Lemma that is of independent interest states that by using a random restriction we can convert an AND of small ORs to an OR of small ANDs and conversely.

1. INTRODUCTION

Proving lower bounds for the resources needed to compute certain functions is one of the most interesting branches of theoretical

Advances in Computing Research, Volume 5, pages 143–170.

ISBN: 0-89232-896-7

computer science. One of the ultimate goals of this branch is of course to show that $P \neq NP$. However, it seems that we are yet quite far from achieving this goal and that new techniques have to be developed before we can make significant progress toward resolving this question. To gain understanding of the problem of proving lower bounds and develop techniques, several restricted models of computation have been studied. Recently there has been significant progress in proving lower bounds in two circuit models. The first example is the case of monotone circuits, i.e., circuits just containing AND and OR gates and no negations. Superpolynomial lower bounds were proved for the clique function by Razborov [R] and these were improved to exponential lower bounds by Alon and Boppana [AB]. Andreev [An] independently obtained exponential lower bounds for other NP functions.

The second model where interesting lower bounds have been proved is the model of *small depth circuits*. These circuits have the full instruction set of AND, OR, and negations and furthermore each AND and OR gate can have an arbitrary number of inputs. However, the depth (the longest path from input to output) is restricted to be small, e.g., constant. The unrestricted size of the AND gates is needed to make it possible to compute circuits depending on all inputs. In this paper we will prove exponential lower bounds for this model. Our technique enables us to prove lower bounds for several different functions. Thus we have at least partial understanding of what causes a function to be difficult to compute in this model of computation.

Finally let us remark that even though the $P \neq NP$ question is one of the motivations to studying the problem of small depth circuits, we do not think that the techniques of this paper will help in resolving that question. The results for small depth circuits and monotone circuits show only that it is possible to prove exponential lower bounds in nontrivial cases. This might be taken as a promising sign and encourage us to look for new techniques with renewed optimism.

1.1. Lower Bounds for Small Depth Circuits; A Crucial Lemma

The problem of proving lower bounds for small depth circuits has attracted the attention of several researchers in the field. Functions considered have been simple functions like parity and majority. The first superpolynomial lower bounds for the circuits

computing parity was obtained by Furst, Saxe, and Sipser [FSS]. Ajtai [Aj] independently gave slightly stronger bounds and Yao [Y] proved the first exponential lower bounds. (The case of monotone small depth circuits has been studied by Boppana [B], Valiant [V], and Klawe, Paul, Pippenger, and Yannakakis [KPPY].)

We will in this paper give almost optimal lower bounds for the size of circuits computing parity. However, it is quite likely that the longer lasting contribution will be our main Lemma. The main lemma is the essential ingredient in the proof and it gives some insight into why some problems require large circuits when the depth is small. The lemma tells us that given a depth two circuit, say an AND of small ORs (a gate is small if it has few inputs), then if one gives random values to a randomly selected subset of the variables it is possible to write the resulting induced function as an OR of small ANDs with very high probability. Let us outline how this can be used to prove lower bounds for circuits computing parity.

Given a circuit of constant depth k computing parity we can give random values to some random inputs. The remaining circuit will still compute parity (or the negation of parity) of the remaining variables. By the virtue of the lemma it is possible to interchange two adjacent levels of ANDs and ORs, then by merging the two now adjacent levels with the same connective decrease the depth of the circuit to $k - 1$. This can be done without increasing the size of the circuit significantly. An easy induction now gives the result.

The idea of giving random values to some of the variables was first introduced in [FSS] and weaker versions of our main lemma were used in [FSS] and [Y]. In [FSS] the probability of the size not increasing too much was not proved to be exponentially small. Yao only proved that the resulting OR of small ANDs was in a technical sense a good approximation of the original function. This fact gave significant complications to the rest of the proof. Also, Yao did not obtain the sharp estimates for the probability of failure. Since we get almost optimal lower bounds for the size of parity circuits our estimates are sharp up to a constant.

1.2. Results

Our nearly optimal results for the size of parity circuits imply that a polynomial size circuit computing parity has to have depth essentially $\log n/\log\log n$. The best previous lower bounds for the

depth of polynomial size parity circuits was $\sqrt{\log n}$ by Ajtai [Aj]. Here as everywhere else in the paper $\log n$ denotes logarithms to base 2.

By similar methods it is possible to prove that there is a family of functions f_k^n of n inputs that have linear size circuits of depth k but require size circuits when restricted to depth $k-1$. These functions f_k^n were introduced by Sipser in [S]. Sipser proved super-polynomial lower bounds for the size of the circuits when the depth was restricted to be $k-1$. Yao claimed exponential lower bounds for the same situation.

1.3. Small Depth Circuits and Relativized Complexity

Lower bounds for small depth circuits have some interesting applications to relativized complexity. Furst, Saxe and Sipser proved in [FSS] that subexponential lower bounds [more precisely $\Omega(2^{(\log n)^i})$ for all i] for any constant depth k for the parity function would imply the existence of an oracle separating PSPACE from the polynomial time hierarchy. Yao [Y] was the first to prove sufficiently good lower bounds to obtain the separation for an oracle A. Cai [C] extended his methods to prove that a random oracle separated the two complexity classes with probability 1.

In [S] Sipser proved the corresponding theorem that the same lower bounds for the functions f_k^n would imply the existence of oracles separating the different levels in the polynomial hierarchy. The lower bounds claimed by Yao gives the first oracle achieving this separation. Our bounds are of course also sufficient. The question whether a random oracle achieves this separation is still open.

1.4. Relations to PRAMs

The model of small depth circuits has relations to computation by parallel random access machines (PRAM). In particular, Stockmeyer and Vishkin [SV] proved that any function that can be computed on a slightly limited PRAM with a polynomial number of processors in time T can also be computed by polynomial size unbounded fanin circuits of depth $O(T)$. The limitations on the PRAM was a limitation of the instruction set to contain only relatively simple operations like addition, comparison, indirect addressing multiplication by $\log n$ size numbers, etc.

Thus, our results imply among other things that parity requires time $\Omega(\log n/\log\log n)$ to compute on such a PRAM. Interestingly enough the same bounds can be proved for more powerful PRAMs using extensions of the present technique [BeH].

1.5. Outline of Paper

In Section 3 we prove the main lemma. The necessary background and some motivation are given in Section 2. The application to parity circuits is in Section 4 and in Section 5 we prove the lower bounds for the functions f_k^n and in Section 6 we briefly mention some more details of the implications for relativized complexity. An earlier version of this paper appeared in [H1]. The paper is also part of my thesis [H2].

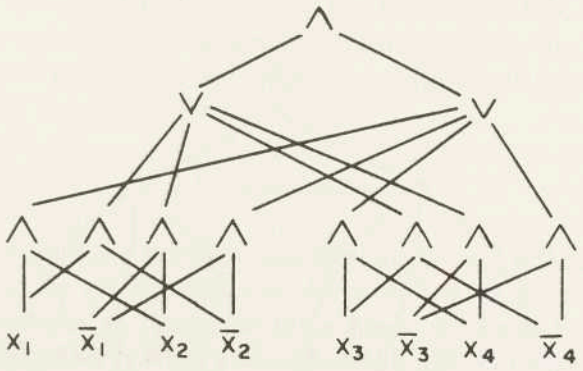
2. BACKGROUND

2.1. Computational Model

We will be working with unbounded fanin circuits of small depth. A typical example looks like Figure 1.

We can assume that the only negations occur as negated input variables. In general if there are negations higher up in the circuit we can move them down to the inputs using DeMorgan's laws. This procedure at most doubles the size of the circuit. Observe that we have alternating levels of AND and OR gates since two adjacent gates of the same type can be collapsed into one gate.

Figure 1.



The crucial parameters for a circuit are the depth and the size. *Depth* is defined as the length of the longest path from an input to the output and can also be thought of as the number of levels of gates. For instance the depth of the circuit in Figure 1 is 3. *Size* is defined to be the total number of AND/OR gates and the circuit in Figure 1 is of size 11. The *fanin* of a gate is defined as the number of inputs to it. We put no restriction on the fanin of the gates in our circuits. However, we will be interested in the *bottom fanin*, which is defined as the maximum fanin for any gate on the lowest level, i.e., having variables as inputs.

2.2. Outline of Proof

Several of the cited lower bounds proofs ([FSS], [Y] and the present paper) have the same outline. The proofs are by induction which proceeds as follows.

1. Prove that parity circuits of depth 2 are large.
2. Prove that small depth k parity circuits can be converted to small depth $k-1$ parity circuits.

Of these two steps the first step is easy and tailored for the parity function. It is easily seen that depth 2 parity circuits require size 2^{n-1} [FSS]. The second step is much more difficult and contains the difference between the papers. The basic idea for doing this lies in the fact that every function can be written either as an AND or ORs or as an OR and ANDs. To give an idea of (2) assume that $k = 3$ and we have the depth 3 circuit shown in Figure 2. Take any gate at distance two from the inputs. It represents a subcircuit of depth 2. In this case this circuit will be an AND of ORs. Now observe that

Figure 2.

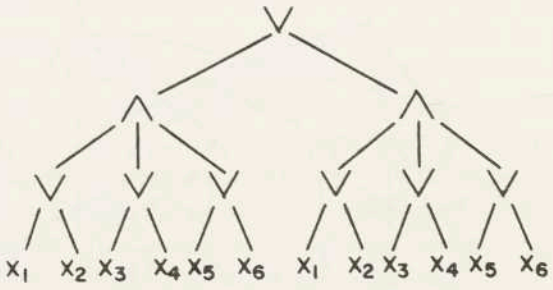


Figure 3.

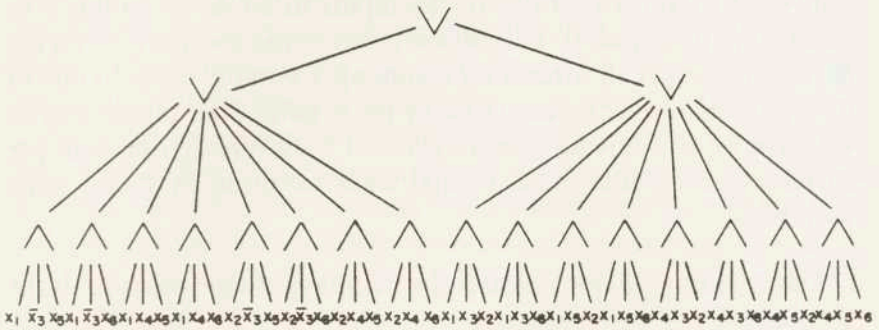
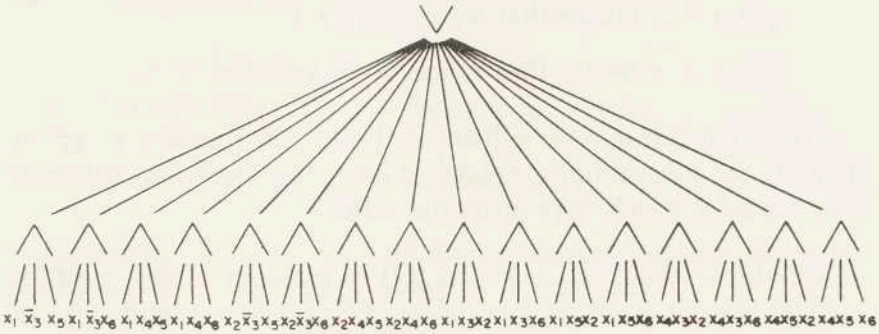


Figure 4.



any function can be written either as an AND of ORs or as an OR of ANDs. Thus, we can change this depth 2 circuit to and OR of ANDs which computes the same function, as shown in Figure 3. Observe that we have two adjacent levels consisting of OR gates. These two levels can be merged to one level and we get the circuit of depth 2 shown in Figure 4. However, doing this we run into one problem. When we convert and AND of ORs to an OR of ANDs the size of the circuit will in general increase considerably. Thus, we have converted a *small* depth k circuit to a *large* depth $k-1$ circuit and hence we fail to achieve (2).

2.3. Restrictions

The way around this problem was introduced in [FSS] and works as follows. If we assign values to some of the variables we can simplify the circuit. In particular if we assign the value 1 to one of the inputs of an OR gate we know that the output of that OR gate

will be 1 no matter what the other inputs are. In the same way we only need to know that one of the inputs to an AND gate is 0 to decide that it outputs 0. This means that for any specific gate on the bottom level we can force it by assigning a suitable value to one of its inputs. However there are many more gates than inputs and so we have to do something more efficient than forcing one gate per variable. Let us first make formal what we mean by fixing some variables.

DEFINITION. A restriction ρ is a mapping of the variables to the set $\{0, 1, *\}$.

$$\rho(x_i) = 0 \text{ means that we substitute the value 0 for } x_i$$

$$\rho(x_i) = 1 \text{ means that we substitute 1}$$

$$\rho(x_i) = * \text{ means that } x_i \text{ remains a variable.}$$

Given a function F we will denote by $F\lceil_\rho$ the function we get by making the substitutions prescribed by ρ. $F\lceil_\rho$ will be a function of the variables which were given the value $*$.

EXAMPLE. Let $F(x_1, x_2, x_3, x_4, x_5) =$ majority of the variables and let $\rho(x_1) = 1, \rho(x_2) = *, \rho(x_3) = *, \rho(x_4) = 1$, and $\rho(x_5) = *$. Then $F\lceil_\rho(x_2, x_3, x_5) =$ "at least one of x_2, x_3 and x_5 is 1."

A simple observation that is important to the proof of the result for parity is:

OBSERVATION. Parity $\lceil_\rho$ = Parity or the negation of Parity.

We are looking for restrictions that simplify circuits efficiently. It seems hard to do this explicitly and we will use a probabilistic method. We will be working with random restrictions with distributions parameterized by a real number p which will usually be small.

DEFINITION. A random restriction $\rho \in R_p$ satisfies

$$\rho(x_i) = 0 \text{ with probability } (1/2) - (p/2)$$

$$\rho(x_i) = 1 \text{ with probability } (1/2) - (p/2)$$

$$\rho(x_i) = * \text{ with probability } p$$

independently for different x_i.

Observe that we have probability p of keeping a variable. Thus, the expected number of variables remaining is pn. Obviously the smaller p is the more we can simplify our circuits but on the other hand we have fewer remaining variables. We have to optimize his trade off when we make a choice of p.

The main improvement of the present paper over previous papers is that we analyze in a better way how much a restriction simplifies a circuit. We will prove a lemma that basically tells us that if we hit a depth 2 circuit with a random restriction then we can change an AND of ORs to an OR of ANDs without increasing the size. We prove that this fails with only exponentially small probability.

We will need some notation. A *minterm* is a minimal way to make a function 1. We will think of a minterm σ for a function F as a partial assignment with the following two properties.

1. σ forces F to be true.
2. No subassignment of σ forces F to be true.

Thus (2) says that σ is minimal satisfying (1).

EXAMPLE. Let $F(x_1, x_2, x_3)$ be the majority function. Then the minterms are σ_1, σ_2, and σ_3 where

$$\sigma_1(x_1) = 1, \quad \sigma_1(x_2) = 1, \quad \sigma_1(x_3) = *$$

$$\sigma_2(x_1) = 1, \quad \sigma_2(x_2) = *, \quad \sigma_2(x_3) = 1$$

$$\sigma_3(x_1) = *, \quad \sigma_3(x_2) = 1, \quad \sigma_3(x_3) = 1.$$

The size of a minterm is defined as the number of variables to which it gives either the value 0 or the value 1. All three of the above minterms are of size 2. Observe that it is possible to write a function as an OR of ANDs where the ANDs precisely correspond to its minterms. The size of the ANDs will be the size of the minterms since x_i will be input precisely when $\sigma(x_i) = 1$ and $\bar{x}_i$ will be input precisely when $\sigma(x_i) = 0$.

3. MAIN LEMMA

Our Main Lemma will tell us that if we apply a restriction we can with high probability convert an AND of ORs to an OR of ANDs.

This will provide the tool for us to carry through the outline of the proof described in Section 2.

MAIN LEMMA. *Let G be an AND of ORs all of size $\leqslant t$ and ρ a random restriction from R_p. Then the probability that $G\lceil_\rho$ cannot be written as an OR of ANDs all of size $< s$ is bounded by α^s where α is the unique positive root to the equation.*

$$\left(1+\frac{4p}{1+p}\frac{1}{\alpha}\right)^t = \left(1+\frac{2p}{1+p}\frac{1}{\alpha}\right)^t + 1.$$

REMARK 1. An elementary argument shows that $\alpha = 2pt/\ln\phi + O(p^2t) < 5pt$, for sufficiently small p, where ϕ is the golden ratio.

REMARK 2. By looking at $\neg G$ one can see that it is possible to convert an OR of ANDs to an AND of ORs with the same probability.

REMARK 3. There are two versions of the proof of the Main Lemma that are identical except for notation. Our original proof was in terms of a labeling algorithm as in those used by Yao [Y] in his proof. The present version of the proof, avoiding the use of such an algorithm, was proposed by Ravi Boppana.

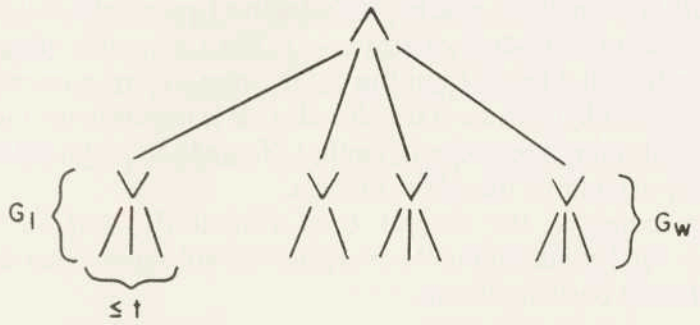
It turns out that it is easier to prove a slightly stronger version of the Main Lemma. First we will require all minterms of $G\lceil_\rho$ to be small. By the remark above this implies that $G\lceil_\rho$ can be written as an OR of small ANDs. A more significant difference between the Main Lemmma and the stronger lemma we will prove is that we will estimate the probability conditioned upon any function being forced to be 1. The reason for this is that it facilitates induction.

For notational convenience let $min(G) \geqslant s$ denote the event that $G\lceil_\rho$ has a minterm of size at least s.

STRONGER MAIN LEMMA. *Let $G = \bigwedge_{i=1}^{w} G_i$, where G_i are OR's of fanin $\leqslant t$. Let F be an arbitrary function. Let ρ be a random restriction in R_p. Then we have*

$$Pr[min(G) \geqslant s \mid F\lceil_\rho \equiv 1] \leqslant \alpha^s.$$

Figure 5.



REMARK 4. The Stronger Main Lemma implies the Main Lemma by choosing $F \equiv 1$ and the fact that a function has a circuit which is an OR of ANDs corresponding to its minterms.

REMARK 5. If there is no restriction ρ satisfying the condition $F\lceil_\rho \equiv 1$ we will use the convention that the conditional probability in question is 0.

Proof. We will prove the Stronger Main Lemma by induction on w, the number of ORs in our depth 2 circuit. A picture of G which is good to keep in mind is shown in Figure 5.

If $w = 0$ the lemma is obvious ($G \equiv 1$). Suppose now that the statement is true for all values less than w. We will show that it is true for w. We will first study what happens to G_1, the first OR in our circuit. We have two possibilities, either it is forced to be 1 or it is not. We will estimate these two probabilities separately. We have

$$Pr[min(G) \geq s | F\lceil_\rho \equiv 1]$$
$$\leq \max(Pr[min(G) \geq s | F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \equiv 1],$$
$$Pr[min(G) \geq s | F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1])$$

The first term is

$$Pr[min(G) \geq s | (F \wedge G_1)\lceil_\rho \equiv 1].$$

However, in this case $G\lceil_\rho = \bigwedge_{i=1}^{w} G_i\lceil_\rho = \bigwedge_{i=2}^{w} G_i\lceil_\rho$ since we are only concerned about ρ's that force G_1 to be 1. Thus $min(G) \geq s$ is

equivalent to saying that $\bigwedge_{i=2}^{w} G_i\lceil_\rho$ has a minterm of size at least s. But this probability is $\leqslant \alpha^s$ by the inductive hypothesis since we are talking about a product of size $w-1$. We are conditioning upon another function being 1 but this is OK since we are assuming that the induction hypothesis is true for all F. It is precisely the fact that the conditioning keeps changing that "forced" us to introduce the stronger version of the Main Lemma.

Now consider the second term $(Pr[min(G) \geqslant s | F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1])$. For notational convenience we will assume that G_1 is an OR of only positive literals, i.e.

$$G_1 = \bigvee_{i \in T} x_i$$

where $|T| \leqslant t$. We do not lose generality by this since ρ is symmetric with respect to 0 and 1 and hence we can interchange x_i and $\bar{x}_i$ if necessary. Let $\rho = \rho_1\rho_2$, where ρ_1 is the restriction of the variables in T and ρ_2 is the restriction of the other variables. Thus, the condition that $G_1\lceil_\rho \not\equiv 1$ is equivalent to that ρ_1 does not take the value 1, and we write the condition as $G_1\lceil_\rho \not\equiv 1$. Since we are now conditioning upon the fact that G_1 is not made true by the restriction, we know that G_1 has to be made true by every minterm of $G\lceil_\rho$, i.e., for every minterm σ there must be an $i \in T$ such that $\sigma(x_i) = 1$. Observe that σ might give values to some other variables in T and that these values might be both 0 and 1. We will partition the minterms of $G\lceil_\rho$ according to what variables in T they give values to. We will call a typical such subset Y.

The fact that the minterm give values to the variables in Y implies in particular that the variables in Y were left as variables and hence were given the value $*$ by ρ_1. We will denote this fact by the shorthand notation $\rho_1(Y) = *$. Further let $min(G)^Y \geqslant s$ denote the event that $G\lceil_\rho$ has a minterm of size at least s whose restriction to the variables in T assigns values to precisely those variables in Y. Using this notation we get

$$\begin{aligned}
&Pr[min(G) \geqslant s | F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1] \\
&\leqslant \sum_{Y \subset T, Y \neq \emptyset} Pr[min(G)^Y \geqslant s | F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1] \\
&= \sum_{Y \subset T, Y \neq \emptyset} Pr[min(G)^Y \geqslant s \wedge \rho_1(Y) = * | F\lceil_\rho \equiv 1 \wedge G_1\lceil_{\rho_1} \not\equiv 1] \\
&= \sum_{Y \subset T, Y \neq \emptyset} Pr[\rho_1(Y) = * | F\lceil_\rho \equiv 1 \wedge G_1\lceil_{\rho_1} \not\equiv 1] \\
&\quad \times Pr[min(G)^Y \geqslant s | F\lceil_\rho \equiv 1 \wedge G_1\lceil_{\rho_1} \not\equiv 1 \wedge \rho_1(Y) = *].
\end{aligned}$$

The inequality and the first equality follows by the reasoning above and the last equality follows by the definition of conditional probability. Now we will estimate each of the two factors in each term of the above sum. Let us start with the first factor (i.e., $Pr[\rho_1(Y) = *|\cdots]$).

To make things simpler we will start by ignoring the condition $F\lceil_\rho \equiv 1$.

LEMMA 1. $Pr[\rho_1(Y) = *|G_1\lceil_\rho \not\equiv 1] = [2p/(1+p)]^{|Y|}$.

Proof. As remarked above the condition $G_1\lceil_\rho \not\equiv 1$ is precisely equivalent to $\rho_1(x_i) \in \{0, *\}$ for $i \in T$. The induced probabilities are $Pr[\rho(x_i) = 0] = (1-p)/(1+p)$ and $Pr[\rho(x_i) = *] = 2p/(1+p)$. The lemma follows since the probabilities are independent. □

Now we have to take the condition $F\lceil_\rho \equiv 1$ into account. The intuition for doing this works as follows. The fact that something is determined to be 1 cannot make stars more likely since having a lot of stars is in a vague sense equivalent to making things undetermined. During a presentation of this material Mike Saks found a nice way to make this formal without looking at probabilities of individual restrictions. We first need an elementary fact from probability theory. Let A, B, and C be three arbitrary events

LEMMA 2. $Pr[A|B \wedge C] \leqslant Pr[A|C]$ *is equivalent to* $Pr[B|A \wedge C] \leqslant Pr[B|C]$.

This lemma follows from use of definition of conditional probability and trivial algebra. Our final estimate will be

LEMMA 3. $Pr[\rho_1(Y) = *|F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1] \leqslant [2p/(1+p)]^{|Y|}$.

Proof. Let $A = [\rho_1(Y) = *]$, $B = (F\lceil_\rho \equiv 1)$, and $C = (G_1\lceil_{\rho_1} \not\equiv 1)$. By the above lemmas we only have to verify that

$$Pr[F\lceil_\rho \equiv 1|\rho_1(Y) = * \wedge G_1\lceil_{\rho_1} \not\equiv 1] \leqslant Pr[F\lceil_\rho \equiv 1|G_1\lceil_{\rho_1} \not\equiv 1].$$

This is clear from inspection since requiring that some variables are $*$ cannot increase the probability that a function is determined. □

Next we try to estimate the other factor. Namely,

$$Pr[min(G)^Y \geqslant s|F\lceil_\rho \equiv 1 \wedge G_1\lceil_{\rho_1} \not\equiv 1 \wedge \rho_1(Y) = *].$$

To do this think of the minterm as consisting of two parts:

1. A part σ_1 that assigns values to the variables of Y.
2. A part σ_2 that assigns values to some variables in the complement $\bar{T}$ of T.

This partition of the minterm is possible since we are assuming that it assign no values to variables in $T - Y$. Observe that σ_2 is a minterm of the function $G\lceil_{\rho\sigma_1}$. This obviously suggests that we can use the induction hypothesis. We only have to get rid of the unpleasant condition that $G_1\lceil_{\rho_1} \not\equiv 1$. This we do by maximizing over all ρ_1 satisfying this condition. We have

$$Pr[min(G)^Y \geqslant s | F\lceil_\rho \equiv 1 \wedge G_1\lceil_{\rho_1} \not\equiv 1 \wedge \rho_1(Y) = *]$$

$$\leqslant \sum_{\sigma_1 \in \{0,1\}^{|Y|}} \Big[\max_{\rho_1(Y)=*, \rho_1(T) \in \{0,*\}^{|T|}} Pr_{\rho_2}[min(G)^{Y,\sigma_1} \geqslant s | (F\lceil_{\rho_1\sigma_1})\lceil_{\rho_2} \equiv 1]\Big].$$

The two last conditions have disappeared because they involve only ρ_1 and we are now interested in a probability over ρ_2. By (2) above we know that $min(G)^{Y,\sigma_1} \geqslant s$ implies that $(G\lceil_{\rho_1\sigma_1})\lceil_{\rho_2}$ has a minterm of size at least $s - |Y|$ on the variables in $\bar{T}$. Thus we can estimate the probability by $\alpha^{s-|Y|}$ using the induction hypothesis.

We have to be slightly careful when we use the induction hypothesis since $G\lceil_{\rho_1\sigma_1}$ might depend on variables in $T - Y$. These variables cannot, by the definition of Y, be in the minterm we are looking for. This implies that we can instead look at the formula $\bigwedge_{\sigma_T} G\lceil_{\rho_1\sigma_1\sigma_T}$ where σ_T ranges over all ways to give values to the remaining variables in T. This is just equivalent to dropping the variables from $T - Y$ in all the G_i. In particular this implies that the number of ORs in the resulting formula is at most $w - 1$ and the induction hypothesis can be applied.

To sum up each term in the sum is estimated by $\alpha^{s-|Y|}$ and we have $2^{|Y|} - 1$ possibly σ_1. This is because σ_1 must make G_1 true and hence cannot be all 0. Thus we get the total bound $(2^{|Y|} - 1)\alpha^{s-|Y|}$.

Finally we must evaluate the sum. Since the term corresponding to $Y = \varnothing$ is 0 we can include it.

$$\sum_{Y \subset T} \left(\frac{2p}{1+p}\right)^{|Y|} (2^{|Y|} - 1)\alpha^{s-|Y|}$$

$$= \alpha^s \sum_{i=0}^{|T|} \binom{|T|}{i} \left[\left(\frac{4p}{1+p}\frac{1}{\alpha}\right)^i - \left(\frac{2p}{1+p}\frac{1}{\alpha}\right)^i\right]$$

$$= \alpha^s \left[\left(1 + \frac{4p}{1+p} \frac{1}{\alpha}\right)^{|T|} - \left(1 + \frac{2p}{1+p} \frac{1}{\alpha}\right)^{|T|} \right]$$

$$\leqslant \alpha^s \left[\left(1 + \frac{4p}{1+p} \frac{1}{\alpha}\right)^{t} - \left(1 + \frac{2p}{1+p} \frac{1}{\alpha}\right)^{t} \right] = \alpha^s.$$

The last equality follows by the definition of α. This finishes the induction step and the proof of the Stronger Main Lemma.

4. LOWER BOUNDS FOR SMALL DEPTH CIRCUITS

The first function we will prove lower bounds for is parity. We have

THEOREM 1. *There are no depth k parity circuits of size $2^{(1/10)^{k/(k-1)} n^{1/(k-1)}}$ for $n > n_0^k$ for some absolute constant n_0.*

REMARK 6. Observe that this is quite close to optimal since it is known that parity can be computed by depth k circuits of size $n2^{[1/(k-1)]}$. The best previous lower bounds were $\Omega(2^{n^{1/4k}})$ by Yao [Y].

As in the case of the Main Lemma it will be more convenient to first prove something that is more suitable to induction.

THEOREM 2. *Parity cannot be computed by a depth k circuit containing $\leqslant 2^{(1/10)n^{1/(k-1)}}$ subcircuits of depth at least 2 and bottom fanin $\leqslant \frac{1}{10} n^{1/(k-1)}$ for $n > n_0^k$ for some absolute constant n_0.*

Proof. We will prove the theorem by induction over k. The base case $k = 2$ follows from the well-known fact that depth 2 parity circuits must have bottom fanin n. The induction step will be done as outlined in Section 2. We can now with the help of the Main Lemma make sure that we convert a *small* depth k circuit to a *small* depth $k - 1$ circuit.

Suppose without loss of generality that our depth k circuits are such that the gates at distance 2 from the inputs are AND gates and hence represent a depth 2 circuit with bottom fanin bounded by $\frac{1}{10} n^{1/(k-1)}$. Apply a random restriction from R_p with $p = n^{-1/(k-1)}$. Then by our lemma every individual depth 2 subcircuit can be written as an OR of ANDs of size bounded by s with probability $1 - \alpha^s$. By the chosen parameters α is bounded by a constant less than $\frac{1}{2}$. If we choose $s = \frac{1}{10} n^{1/(k-1)}$ the probability that any of the $2^{(1/10)n^{1/(k-1)}}$ depth 2 circuits cannot be converted into a depth 2 circuit of the other type is bounded by $(2\alpha)^s$. Thus, with probability at least

$1-(2\alpha)^s$ we can interchange the order of AND and OR in all depth 2 subcircuits and still have bottom fanin bounded by s. Observe that this gives us two adjacent levels of OR's that can be collapsed to decrease the depth of the circuit to $k-1$.

The number of remaining variables is expected to be $n^{(k-2)/(k-1)}$ and with probability greater than $\frac{1}{3}$ we will get at least this number. Thus with nonzero probability we can interchange the order of AND and OR in all depth 2 circuits and we also have at least $n^{(k-2)/(k-1)}$ remaining variables. In particular such a restriction exists. Applying this restriction to the circuit gives a depth $k-1$ circuit computing parity of at least $n^{(k-2)/(k-1)} = m$ variables. Further it has bottom fanin bounded by $\frac{1}{10}n^{1/(k-1)} = \frac{1}{10}m^{1/(k-2)}$ and the number of gates of depth at least 2 is bounded by $2^{1/10}n^{1/(k-1)} = 2^{1/10}m^{1/(k-2)}$. The last fact follows because a gate of depth at least 2 in the new circuit corresponds to a gate of depth at least three in the old depth k circuit. But this is precisely a circuit that is certified not to exist by the induction hypothesis. The proof of Theorem 2 is complete. □

Let us now prove Theorem 1. Consider the circuit as a depth $k+1$ circuit with bottom fanin 1. Hit it with a restriction from R_p using $p = \frac{1}{10}$ and by using our Main Lemma with $s = \frac{1}{10}n^{1/(k-1)}$ we see that we get a circuit which does not exist by Theorem 2.

Since there are no constants depending on k hidden in the theorem we get the following corollary.

COROLLARY. *Polynomial size parity circuits must have depth at least* $\log n/(c+\log\log n)$ *for some constant* c.

Observe that this is tight since for every constant c there are such polynomial size circuits. Since Yao had constants in his theorems it is not clear if similar corollaries could be obtained from [Y].

Observe that we have used very little about parity. Only the lower bound for $k=2$ and the fact that it behaves well with respect to restrictions. Thus, we will be able to improve lower bounds for sizes of small depth circuits for other functions using our Main Lemma. Let us do majority:

THEOREM 3. *Majority requires size* $2^{(1/10)^{k/(k-1)}n^{1/(k-1)}}$ *depth* k *circuits for* $n > n_0^k$ *for some absolute constant* n_0.

Proof. To make the proof go through we only need to make two observations. The base case $k = 2$ goes through. Second, even if we require that the restriction gives out as many 1's as 0's we still have a nonzero probability that a random restriction satisfies all conditions. This requirement ensures that the smaller circuit also computes majority.

In general we do not need that we get back the same function but only that we get a function that is hard to compute. Loosely speaking we can prove the corresponding lower bounds as soon as the function even when hit by severe restriction still have large minterms. We leave the details to the interested reader.

5. FUNCTIONS REQUIRING DEPTH k TO HAVE SMALL CIRCUITS

We prove that there are functions f_k^m that have linear size circuits of depth k but require exponential size circuits when the depth is restricted to $k - 1$. To prove this we will introduce a new probability space of restrictions and reprove the Main Lemma for this space of restrictions.

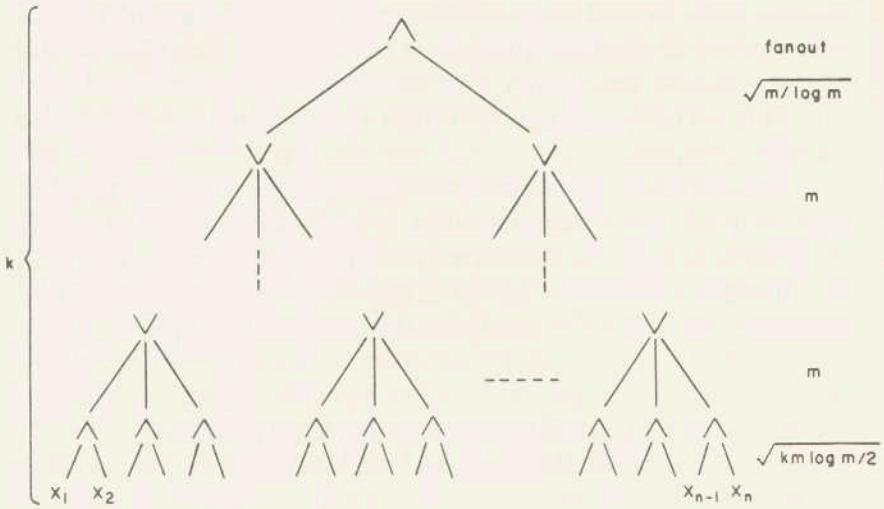
5.1. The Sipser Functions f_k^m

In [Si], Sipser defined a set of functions f_k^m that could be computed in depth k and linear size. He showed, however, that these functions require superpolynomial size when the depth is restricted to $k - 1$. We will redefine f_k^m slightly and let it denote the function defined by the circuit in Figure 6. To avoid confusion we will refer to the circuit in Figure 6 as the defining circuit of f_k^m. The defining circuit is thus a tree with top fanin $\sqrt{m/\log m}$, bottom fanin $\sqrt{km \log m/2}$, while all the other fanouts are m. Each variable occurs at only one leaf. Thus, by definition f_k^m is a function of $m^{k-1}\sqrt{k/2}$ variables.

Yao has claimed exponentially lower bounds for these functions, but the proof has not yet appeared. We have the following results for the functions f_k^m.

THEOREM 4. *Depth $k - 1$ circuits computing f_k^m are of size at least $2^{(1/12\sqrt{2k})\sqrt{m/\log m}}$ for $m > m_1$, where m_1 is some absolute constant.*

Figure 6.



As an immediate corollary we get

COROLLARY. *Polynomial size circuits of depth* $f(n)$ *are more powerful than polynomial size circuits of depth* $f(n) - 1$ *if* $f(n) < (\log n/3 \log\log n) - \omega[\log n/(\log\log n)^2]$.

Proof. Follows from a computation using $n = m^{k-1}\sqrt{k/2}$ and $k = f(n)$.

5.2. New Random Restrictions

One would like to prove Theorem 4 with the aid of the Main Lemma. Here, however, one runs into problems not encountered in the case of the parity function. If a restriction from R_p is applied to f_k^m the resulting function will be a constant function with very high probability. This happens since the gates at the bottom level are quite wide and with very high probability all gates will be forced. There is also a more philosophical reason why R_p destroys functions like f_k^m. R_p was designed to destroy any small-depth circuit, and will in particular destroy the circuits defining f_k^m. To get around this problem we will define another set of restrictions that are designed not to destroy the circuits defining f_k^m.

DEFINITION. Let q be a real number and $(B_i)_{i=1}^r$ a partition of the variables (The B_i are disjoint sets of variables and their union is the set of all variables.) Let $R_{q,B}^+$ be the probability space of restrictions that takes values as follows.

For $\rho \in R_{q,B}^+$ and every B_i, $1 \leqslant i \leqslant r$ independently:

1. With probability q let $s_i = *$ and else $s_i = 0$.
2. For every $x_k \in B_i$ let $\rho(x_k) = s_i$ with probability q and else $\rho(x_k) = 1$.

Similarly a $R_{q,B}^-$ probability space of restrictions is defined by interchanging the roles played by 0 and 1.

The idea behind these restrictions is that a block B_i will correspond to the variables leading into one of the ANDs in the bottom level in the circuit defining f_k^m. If the bottom level gates are ORs we use a restriction from R_q^-. These restrictions will, however, not be quite sufficient for our purposes and we need a complementary restriction.

DEFINITION. For a restriction $\rho \in R_{q,B}^+$ let $g(\rho)$ be a restriction defined as follows: For all B_i with $s_i = *$, $g(\rho)$ gives the value 1 to all variables given the value $*$ by ρ except one to which it gives the value $*$. To make $g(\rho)$ deterministic we assume that it gives the value $*$ to the variable with the highest index given the value $*$ by ρ. If $\rho \in R_{q,B}^-$, then $g(\rho)$ is defined similarly but now takes the values 0 and $*$.

These probability spaces of restrictions do not assign values to variables independently as R_p did, but is nice enough so that the proof of our Main Lemma will go through with only minor modifications. Let $\rho g(\rho)$ denote the composition of the two restrictions. Observe that they are compatible since $g(\rho)$ assigns values to precisely the variables given the value $*$ by ρ.

LEMMA 4. *Let G be an AND of ORs all of size $\leqslant t$ and ρ a random restriction from $R_{q,B}^+$. Then the probability that $G\restriction_{\rho g(\rho)}$ cannot be written as an OR of ANDs all of size $< s$ is bounded by α^s, where* $\alpha = 4q/(2^{1/t} - 1) < 4qt/\log 2 < 6qt$.

REMARK 7. The same is true for $R_{q,B}^-$.

REMARK 8. The probability of converting an OR of ANDs to an AND of ORs is the same.

As in the case of the Main Lemma, before proving Lemma 4, we prove a stronger lemma stating that we have the same estimate of the probability even when we condition upon an arbitrary function being forced to 1 by ρ. Define $AND(G\lceil_{\rho g(\rho)}) \geqslant s$ denote the event that $G\lceil_{\rho g(\rho)}$ cannot be written as an OR of ANDs of size $< s$.

LEMMA 5. *Let $G = \bigwedge_{i=1}^{w} G_i$, where G_i are OR's of fanin $\leqslant t$. Let F be an arbitrary function. Let ρ be a random restriction in $R_{q,B}^{+}$. Then*

$$Pr[AND(G\lceil_{\rho g(\rho)} \geqslant s | F\lceil_{\rho} \equiv 1] \leqslant \alpha^{s}$$

where $\alpha = 4q/(2^{1/t} - 1)$.

REMARK 9. Recall that if there is no restriction ρ satisfying the condition $F\lceil_{\rho} \equiv 1$ then the conditional probability in question is defined to be 0. Observe that we are only conditioning upon $F\lceil_{\rho} \equiv 1$ and not $F\lceil_{\rho g(\rho)} \equiv 1$.

Proof. We will only use the weaker bound $6qt$ for α and since the lemma is trivially true if $\alpha \geqslant 1$ we will assume $q < 1/6t$ whenever convenient. The proof will be done in a way similar to the proof of the Stronger Main Lemma. We therefore only outline the proof, and give details only where the proofs differ.

As before

$$\begin{aligned} &Pr[AND(G\lceil_{\rho g(\rho)} \geqslant s | F\lceil_{\rho} \equiv 1] \\ &\leqslant max(Pr[AND(G\lceil_{\rho g(\rho)} \geqslant s | F\lceil_{\rho} \equiv 1 \wedge G_1\lceil_{\rho} \equiv 1, \\ &\quad Pr[AND(G\lceil_{\rho g(\rho)} \geqslant s | F\lceil_{\rho} \equiv 1 \wedge G_1\lceil_{\rho} \not\equiv 1]). \end{aligned}$$

The first term,

$$Pr[AND(G\lceil_{\rho g(\rho)} \geqslant s | (F \wedge G_1)\lceil_{\rho} \equiv 1]$$

is taken care of by the induction hypothesis.

We have to estimate the second term, $Pr[AND(G\lceil_{\rho g(\rho)} \geqslant s | F\lceil_{\rho} \equiv 1 \wedge G_1\lceil_{\rho} \not\equiv 1]$. We cannot assume that G_1 is an OR of only positive literals since the restrictions employed here assign 0 and 1 nonsymmetrically.

We denote the set of variables occurring in G_1 by T, and $|T| \leqslant t$. We do not know that G_1 must be made true by every minterm of $G\lceil_{\rho g(\rho)}$. This is because G_1 might be made true by $g(\rho)$. We do know, however, that for $G\lceil_\rho$ not to be the constant 0 some of the variables of T must be given the value $*$ by ρ. Suppose the variables of T belong to r different blocks. Assume for notational convenience that these blocks are B_i, $i = 1, \ldots, r$. We call a block B *exposed* if there is a variable $x_i \in B$ such that $x_i \in T$ and $\rho(x_i) = *$. By the above remark there must be some exposed blocks for G not identically 0. Let Y denote the set of exposed blocks. Denote this event by $exp(Y)$ and let $[r]$ denote the set $\{1, 2, \ldots, r\}$. We get

$$\begin{aligned} &Pr[AND(G\lceil_{\rho g(\rho)} \geqslant s \mid F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1] \\ &\leqslant \sum_{Y \subset [r], Y \neq \varnothing} Pr[exp(Y) \mid F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1] \\ &\times Pr[AND(G\lceil_{\rho g(\rho)} \geqslant s \mid F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1 \wedge exp(Y)]. \end{aligned}$$

The factors in the above sum can be estimated separately. Let us start with the first factor $Pr[exp(Y) \mid F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1]$. We need a little bit of extra notation. Let $P_i = \{j \mid x_j \in G_1 \wedge x_j \in B_i\}$ and let $N_i = \{j \mid \bar{x}_j \in G_1 \wedge x_j \in B_i\}$. Let us start with the simple case when Y consists of a single block B_i.

LEMMA 6. $Pr[exp(B_i) \mid F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1] \leqslant 2q$.

Proof. By the definition of conditional probability we want to prove

$$\frac{\sum'_{exp(B_i)} Pr(\rho)}{\sum' Pr(\rho)} \leqslant 2q.$$

Here the prime indicates that we are only summing over ρ satisfying the condition $F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1$. Remember that if this quotient takes the form 0/0 we have the convention that it takes the value 0. Now assume that ρ gives a nonzero contribution to the numerator. We define a restriction $\tilde{\rho} = H(\rho)$, which gives a larger contribution to the denominator. Let

1. $\tilde{\rho}(x_j) = \rho(x_j)$ for $x_j \notin B_i$
2. $\tilde{\rho}(x_j) = 0$ for $j \in P_i$
3. $\tilde{\rho}(x_j) = 1$ for $j \in N_i$
4. $\tilde{\rho}(x_j) = 1$ for $j \in B_i - N_i - P_i$ and $\rho(x_j) = 1$
5. $\tilde{\rho}(x_j) = 0$ for $j \in B_i - N_i - P_i$ and $\rho(x_j) = *$.

To check that $\tilde{\rho}$ gives a contribution to the denominator we only have to check that $F\lceil_{\tilde{\rho}} \equiv 1$ and $G_1\lceil_{\tilde{\rho}} \not\equiv 1$. The first fact follows by noting that we change values only from $*$ to non-$*$ values. To see that the second condition is fulfilled we observe that rules 2 and 3 are tailored to this.

To get an estimate for the quotient we must compute the probability of ρ compared to $\tilde{\rho}$. We must also investigate what restrictions $\bar{\rho}$ satisfy $H(\bar{\rho}) = \tilde{\rho}$. Let us start with this second task.

Observe first that $s_i = *$ in the definition of $\bar{\rho}$ and hence that $\bar{\rho}$ only gives out the values $*$ and 1 on B_i. Obviously $\bar{\rho}(x_j) = \rho(x_j)$ for all x_j not in P_i or N_i. Furthermore $\bar{\rho}(x_j) = *$ for $x_j \in P_i$ since $G\lceil_{\tilde{\rho}} \not\equiv 1$. Finally $\bar{\rho}$ can take any combination of 1 and $*$ on N_i provided it does not take the value of all 1 in the case when P_i is empty. Observe that all these $\bar{\rho}$ might not satisfy the condition $F\lceil_{\tilde{\rho}} \equiv 1$ but we are only trying to get an upper bound. Assume that $\bar{\rho}$ assigns l $*$'s on N_i and $|N_i| - l$ ones. Then

$$Pr(\bar{\rho}) = \frac{q}{1-q} \frac{q^l (1-q)^{|N_i|-l}}{(1-q)^{|N_i|}} Pr(\tilde{\rho}).$$

The first factor comes from the fact that $s_i = 0$ for $\tilde{\rho}$ while $s_i = *$ for $\bar{\rho}$. The second factor comes from the behaviour on N_i. Observe that the probability that $\tilde{\rho}$ gives out only 0 on P_i is equal to the probability that $\bar{\rho}$ gives out only $*$. Summing up we get

$$\sum_{H(\bar{\rho})=\tilde{\rho}} Pr(\bar{\rho}) \leqslant \frac{q}{1-q} Pr(\tilde{\rho}) \sum_{l=0}^{|N_i|} \binom{|N_i|}{l} \left(\frac{q}{1-q}\right)^l$$

$$= \frac{q}{1-q} Pr(\tilde{\rho})(1-q)^{-|N_i|}$$

since $|N_i| \leqslant t$ and $q < 1/6t$ we have $(1-q)^{-|N_i|} < 2$. Using this we have

$$\frac{\sum'_{exp(B_i)} Pr(\rho)}{\sum' Pr(\rho)} \leqslant \frac{\sum'_{\tilde{\rho}} \sum'_{\rho, H(\rho)=\tilde{\rho}, exp(B_i)} Pr(\rho)}{\sum' Pr(\rho)}$$

$$\leqslant \frac{\sum'_{\tilde{\rho}} [2q/(1-q)] Pr(\tilde{\rho})}{\sum'_{\tilde{\rho}} (1 + [2q/(1-q)]) Pr(\tilde{\rho})} \leqslant 2q$$

and the proof is complete. □

Next we have

LEMMA 7. $Pr[exp(Y)|F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1] \leqslant (2q)^{|Y|}$.

Proof. This is proved in the same way as Lemma 6. We get restrictions contributing to the denominator by doing the changes in ρ on all the blocks simultaneously □

Next we estimate the factor

$$Pr[AND(G\lceil_{\rho g(\rho)})^Y \geqslant s|F\lceil_\rho \equiv 1 \wedge G_1\lceil_\rho \not\equiv 1 \wedge exp(Y)].$$

We want to use induction and to do this we have to get rid of the condition $G_1\lceil_\rho \not\equiv 1$. In the blocks that are not exposed we know that ρ only takes the values 0 or 1. This conditioning can be incorporated in $F\lceil_\rho = 1$.

In the exposed blocks we let the corresponding variables that are still alive after $\rho g(\rho)$ be in the ANDs of $G\lceil_{\rho g(\rho)}$. We try all possibilities of these variables and we estimate the probability that the remaining formula cannot be written as an OR of ANDs of size $s - |Y|$. This probability is taken over a restriction that does not include the blocks of Y. Thus, we can use the induction hypothesis and we get the estimate $\alpha^{s-|Y|}$ for each setting of the variables corresponding to Y. Thus we get the total bound $2^{|Y|}\alpha^{s-|Y|}$.

Finally we evaluate the sum to get

$$\sum_{Y\subset[r],Y\neq\varnothing} (2q)^{|Y|}2^{|Y|}\alpha^{s-|Y|} = \alpha^s \sum_{i=1}^{r} \binom{r}{i}\left(\frac{4q}{\alpha}\right)^i$$

$$\leqslant \alpha^s\left[\left(1+\frac{4q}{\alpha}\right)^r - 1\right] = \alpha^s(2^{r/t}-1) \leqslant \alpha^s.$$

This finishes the induction step and the proof of the Lemma 5. □

An interesting question is for what probability distributions on the space of restrictions is it possible to prove the lemma equivalent to the Main Lemma and Lemma 4. The general proof technique uses two crucial properties of the distribution.

1. The condition $F\lceil_\rho \equiv 1$ for an arbitrary F does not bias the value of any variable too much toward $*$. This should also remain true even if we know that the variable is not 1 (0).

2. It is possible to eliminate the variables of G_1 and use induction on a similar restriction over the remaining variables.

Condition (1) was taken care of by Lemmas 3 and 7. Condition (2) seems easier to satisfy and was so obviously satisfied that no formal lemma was needed. The verification was basically done where we claimed that induction could be used after eliminating G_1.

5.3. Back to the Proof of Theorem 4

Let us continue with the present restriction space $R^+_{q,B}$ and prove Theorem 4. We first prove a slightly stronger technical theorem.

THEOREM 5. *There are no depth k circuits computing f^m_k with bottom fanin $(1/12\sqrt{2k})\sqrt{m/\log m}$ and $\leqslant 2^{(1/12\sqrt{2k})\sqrt{m/\log m}}$ gates of depth $\geqslant 2$ for $m > m_0$ some absolute constant m_0.*

Note that Theorem 5 implies Theorem 4 since a depth $k-1$ circuit can be considered as a depth k circuit with bottom fanin 1. Theorem 5 is proved by induction over k. The base case for $k = 2$ is quite easy and is left to the reader.

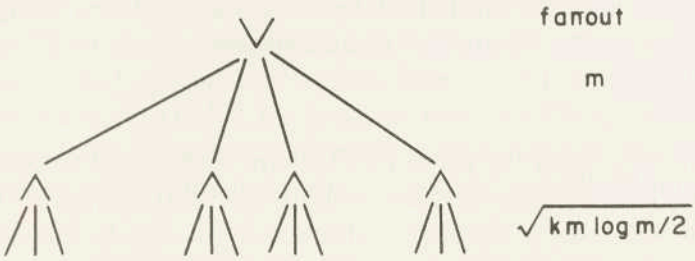
For the induction step we use one of the restrictions defined above. Assume for definiteness that k is odd, so that the gates on the bottom level are AND gates. Define the sets B_i in the partition to be the set of variables leading into an AND gate. Recall that since the defining circuit of f^m_k is a tree the blocks are disjoint. Set $q = \sqrt{2k \log m/m}$ and apply a random restriction from $R^+_{q,B}$.

In the case of the parity function even after applying a restriction, it was trivial that the remaining circuit still computed parity or the negation of parity. In the case of f^m_k, we have to prove that the new restrictions used transform f^m_k into something that is very close to f^m_{k-1}.

LEMMA 8. *If k is odd then the circuit that defines $f^m_k \lceil_{\rho g(\rho)}$ for a random $\rho \in R^+_q$ with $q = \sqrt{2k \log m/m}$, will contain the circuit that defines f^m_{k-1} with probability at least $\frac{2}{3}$, for all m such that $m/\log m \geqslant 100k$, $m > m_1$, where m_1 is some absolute constant.*

REMARK 10. Lemma 8 holds for even k when R^+ is replaced by R^-.

Figure 7.



Proof. The fact that k is odd implies that the two lower levels look like Figure 7.

We establish a series of facts.

Fact 1. The AND gate corresponding to block B_i takes the value s_i for all i, with probability at least $\frac{5}{6}$ for $m > m_0$.

The AND gate corresponding to block B_i takes the values s_i precisely when not only ones are given to the block. The probability of this happening is $(1-q)^{|B_i|} = (1-\sqrt{2k \log m/m})^{\sqrt{km/2 \log m}} < e^{-k \log m} < \frac{1}{6} m^{-k}$ for $m > m_0$. Thus the probability that this happens for any block B_i is bounded by $\frac{1}{6}$ for $m > m_0$.

Fact 2. With probability at least $\frac{5}{6}$ at least $\sqrt{(k-1)m \log m/2}$ inputs given the value $*$ by $\rho g(\rho)$ to each OR gate at level $k-1$. Again this is true only for sufficiently large m.

The expected number of such inputs is $\sqrt{2km \log m}$ and the fact follows from known estimates using $m/\log m \geqslant 100k$. For completeness let us include a very elementary proof.

Let p_i be the probability that an OR gate has an input exactly i AND gates that take the value $*$. Then

$$p_i = \binom{m}{i} \left(\frac{2k \log m}{m}\right)^{1/2} \left(1 - \sqrt{\frac{2k \log m}{m}}\right)^{m-i}.$$

Then for $i < \sqrt{km \log m}$ we have $p_i/p_{i-1} \geqslant \sqrt{2}$. Using $p_{\sqrt{mk \log m}} < 1$ we estimate $\sum_{i=1}^{\sqrt{mk \log m/2}} p_i$ by

$$\sum_{i=1}^{\sqrt{mk \log m/2}} p_i \leqslant p_{\sqrt{mk \log m/2}} \sum_{i=0}^{\infty} 2^{-1/2} \leqslant 4 p_{\sqrt{mk \log m/2}}$$

$$\leqslant 4\sqrt{2}^{-[1-(1/\sqrt{2})]\sqrt{mk \log m}}\, p_{\sqrt{mk \log m}} \leqslant 4\sqrt{2}^{-[1-(1/\sqrt{2})]10k \log m} \leqslant \tfrac{1}{6} m^{-k}$$

for $m > m_0$.

To sum up, with probability at least $\frac{2}{3}$ all OR gates at level $k-2$ will remain undetermined, and have at least $\sqrt{(k-1)m \log m/2}$ variables as inputs. This constitutes the defining circuit for f_{k-1}^m. The lemma is proved. □

Let us now finish the proof of Theorem 5. We need to perform the induction step. This is done using the same argument as in the proof of Theorem 2. Apply a restriction from $R_{q,B}^+$ to the circuit. Observe first that if $m/\log m < 100k$ the result of the theorem is trivial and hence we can assume that the reverse inequality holds. By Lemma 8 the defining circuit still computes a function as difficult as f_{k-1}^m and setting some of the remaining variables the circuit can be made into the defining circuit of f_{k-1}^m.

On the other hand suppose that there existed a circuit of depth k, bottom fanin $(1/12\sqrt{2k})\sqrt{m/\log m}$ and size $2^{(1/12\sqrt{2k})\sqrt{m \log m}}$ which computed f_k^m. By using Lemma 4 and reasoning as in the proof of Theorem 2 we can interchange the ANDs and ORs on the last two levels without increasing the bottom fanin. Now it is possible to collapse two adjacent levels of OR gates and the resulting circuit will be of depth $k-1$. As in the proof of Theorem 2 the gates corresponding to subcircuits of depth 2 in this new circuit correspond to gates of depth 3 in the old circuit. Thus we have obtained a circuit certified not to exist by induction. □

6. SEPARATION OF COMPLEXITY CLASSES BY ORACLES

As mentioned in the introduction lower bound results for small depth circuits can be used to construct oracles relative to which certain complexes are different [FSS], [S]. In particular the result for parity implies that there are oracles for which PSPACE is different from the polynomial time hierarchy. In the same way Theorem 4 implies that there are oracles separating the different levels within the polynomial time hierarchy. As previously remarked, Yao's bounds [Y] were sufficient to obtain these separations. Cai [C] proved that PSPACE was different from the polynomial time hierarchy even for a random oracle. To prove this result one needs to establish that a small circuit makes an error when trying to compute parity on a random input with a probability close to $\frac{1}{2}$. This problem and related problems are studied in [H2].

To prove that a random oracle separates the different levels within the polynomial hierarchy one would have to strengthen Theorem 4 to say that no depth $k-1$ circuit computes a function that agrees with f_k^m for most inputs. This is not true in the case of f_k^m since if k is even (odd), the constant function 1(0) agrees with f_k^m for most inputs. However, perhaps it is possible to get around this by defining other functions more suited to this application.

ACKNOWLEDGMENTS

I am very gateful to Ravi Boppana for reading an early draft of the paper and suggesting the version of the proof avoiding the labeling algorithm. Mike Saks' observation, which simplified the proof of Lemma 3, was also helpful. I am also grateful to several people who have read and commented on drafts of this paper. These people include Ravi Boppana, Zvi Galil, Oded Goldreich, Shafi Goldwasser, Jeff Lagarias, Silvio Micali, Nick Pippenger, and David Shmoys. Supported by an IBM fellowship, partially supported by NSF grant DCR-8509905. Some of the work was done while the author visited AT&T Bell Laboratories.

REFERENCES

[Aj] M. Ajtai, "Σ_1^1-Formulae on finite structures," *Ann. Pure Appl. Logic* 24: 1–48 (1983).

[AB] N. Alon and R. Boppana, "The monotone circuit complexity of Boolean functions," *Combinatorica*, submitted.

[An] A. E. Andreev, "On one method of obtaining lower bounds of individual monotone function complexity," *Dokl. Ak. Nauk.* 282: 1033–1037 (1985).

[BeH] P. Beame and J. Hastad, "Optimal bounds for decision problems on the CRCW PRAM," *Proc. 18th Annu. ACM Symp. Theory Computing*, in press.

[B] R. Boppana, "Threshold functions and bounded depth monotone circuits," *Proc. 16th Annu. ACM Symp. Theory Computing* 475–479 (1984). *J. Computer System Sci.*, in press.

[C] J. Cai, "With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy," *Proc. 18th Annu. ACM Symp. Theory Computing* 21–29 (1986).

[FSS] M. Furst, J. Saxe, and M. Sipser, "Parity, circuits, and the polynomial time hierarchy," *Proc. 22nd Annu. IEEE Symp. Found. Computer Sci.* 260–270 (1981).

[H1] J. Hastad, "Almost optimal lower bounds for small depth circuits," *Proc. 18th Annu. ACM Symp. Theory Computing* 6–20 (1986).

[H2] J. Hastad, *Computational Limitations for Small-Depth Circuits*. MIT Press, 1986.

[KPPY] M. Klawe, W. Paul, N. Pippenger, and M. Yannakakis, "On monotone formulae with restricted depth," *Proc. 16th Annu. ACM Symp. Theory Computing* 480–487 (1984).

[R] A. A. Razborov, "Lower bounds for the monotone complexity of some Boolean functions," *Dokl. Ak. Nauk.* 281: 798–801 (1985).

[S] M. Sipser, "Borel sets and circuit complexity," *Proc. 15th Annu. ACM Symp. Theory Computing* 61–69 (1983).

[SV] L. J. Stockmeyer and U. Vishkin, "Simulation of parallel random access machines by circuits," *SIAM J. Computing* 13(2): 404–422 (1984).

[V] L. Valiant, "Exponential lower bounds for restricted monotone circuits," *Proc. 15th Annu. ACM Symp. Theory Computing* 110–117 (1983).

[Y] A. Yao, "Separating the polynomial-time hierarchy by oracles," *Proc. 26th Annu. IEEE Symp. Found. Computer Sci.* 1–10 (1985).

ANALYSIS OF ERROR CORRECTION BY MAJORITY VOTING

Nicholas Pippenger

ABSTRACT

For a particular model of computation with error correction by majority voting, we determine the reliability required of the processors, the accuracy required of the initial data, and the accuracy attainable in the final result. We determine the minimum possible ratio of correction steps to computation steps and show that, although it has simple asymptotics, it has diabolically complicated local behavior.

1. INTRODUCTION

Our goal in this paper is to analyze a model of computation with error correction by majority voting. In this model, a computation

Advances in Computing Research, Volume 5, pages 171–198.

ISBN: 0-89232-896-7

consists of *steps*, each of which may be either a *computation* step or a *correction* step.

We shall assume not that we have access to the initial data, but only that we have access to an unlimited number of samples of each initial datum, with each sample independently being correct with probability x. (If a sample of an initial datum or an intermediate or final result is correct, we shall say that it is *accurate*. The probability that it is accurate will be called its *accuracy*.)

We shall assume not that each processor performs correctly, but only that each performance of each processor is independently correct with some probability ϱ. (If a processor performs correctly, we shall say that it performs *reliably*. The probability that it performs reliably will be called its *reliability*.)

Suppose that during a computation step, each processor combines three inputs to produce a single output. We assume that the output of a processor is accurate if and only if all three of its inputs are accurate and it performs reliably. (We do not consider the possibility that two or more errors might cancel.) If its inputs are independently accurate with probability x, and if it performs reliabily with probability ϱ, then its output is accurate with probability ϱx^3.

Suppose that during a correction step, each processor takes a majority vote among three independently computed samples of some intermediate result and produces a single sample of this result. We assume that the output sample is accurate if and only if at least two of the three input samples are accurate and if the processor performs reliably. If the inputs are independently accurate with probability x, and if the processor performs reliably with probability ϱ, then the output is accurate with probability $\varrho(3x^2 - 2x^3)$.

Suppose that the samples of the initial data are independently accurate with probability x. Suppose that we wish to produce a final result with accuracy y. For what values of ϱ, x, and y can we perform an unlimited number of computation steps? This question is answered in Section 2, where it is shown that the conditions $\varrho > (4 + 3\sqrt{2})/9 = 0.9158\ldots$,

$$x > \frac{3 - \sqrt{9 - 8/\varrho}}{4}$$

and

$$y < \frac{3 + \sqrt{9 - 8/\varrho}}{4}$$

are necessary and sufficient.

In Section 3 we address the question: what is the minimum number $T_\varrho(G, x, y)$ of correction steps necessary for a given number G of computation steps? We shall show that $T_\varrho(G, x, y)/G$ tends to a limit as $G \to \infty$ that is independent of x and y (subject to the inequalities given above). Let Θ_ϱ denote this limit.

We give an expression for Θ_ϱ in terms of the "rotation number" of a certain homeomorphism of a circle. This expression is not very explicit, but no simple analytic formula for Θ_ϱ exists. Indeed, we show in Section 5 that Θ_ϱ is a diabolically complicated function of ϱ. Although it is continuous and nonincreasing, and its range is the interval $(0, \infty)$, it is locally constant on a dense open set. In particular, every rational number on $(0, \infty)$ is assumed by Θ_ϱ for all ϱ in a closed interval of positive length, and every irrational number in $(0, \infty)$ is assumed by Θ_ϱ for a unique value of ϱ.

Despite these complications, Θ_ϱ has simple asymptotics. In Section 4 we show that it is asymptotic to

$$\frac{2}{\log_3[1/(1 - \varrho)]}$$

as $\varrho \to 1$, and

$$\left(\log \frac{3 + \sqrt{2})}{2}\right)\left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \left(\log \frac{4 - \sqrt{2}}{2}\right)\left(\log \frac{2 + \sqrt{2}}{2}\right)$$

as $\varrho \to \varrho_\infty = (4 + 3\sqrt{2})/9$.

The model analyzed in this paper has its origins in the work of von Neumann [Ne], though we are interested in minimizing the depth rather than the size of networks. To obtain precise information about the depth, we have ignored the size, with the result that it may increase exponentially with the depth. To obtain comparably precise information about the size appears at present to be hopeless. For

related work, see Dobrushin and Ortyukov [DO1, DO2] and Pippenger [P1, P2].

2. LIMITS TO RELIABILITY AND ACCURACY

For $\varrho \in (0, 1)$ and $x \in (0, 1]$, define

$$f_\varrho(x) = \varrho(3x^2 - 2x^3)$$

and

$$g_\varrho(x) = \varrho x^3.$$

We shall say that a function h is *deflationary*, *stationary*, or *inflationary* at x if $h(x) < x$, $h(x) = x$, or $h(x) > x$, respectively. We shall say that h is deflationary or inflationary on X if it is deflationary or inflationary, respectively, at x for all $x \in X$.

For $\varrho \in (0, 1)$, the maps f_ϱ and g_ϱ are increasing on $(0, 1]$, and g_ϱ is deflationary on $(0, 1]$. For $\varrho \in (0, 8/9)$, f_ϱ is also deflationary on $(0, 1]$. For $\varrho = 8/9$, f_ϱ is deflationary on $(0, 3/4)$, stationary at $3/4$, and deflationary on $(3/4, 1]$. For $\varrho \in [8/9, 1)$, define

$$\xi_\varrho^- = \frac{3 - \sqrt{9 - 8/\varrho}}{4}$$

and

$$\xi_\varrho^+ = \frac{3 + \sqrt{9 - 8/\varrho}}{4}.$$

Note that $1/2 < \xi_\varrho^- \leqslant 3/4 \leqslant \xi_\varrho^+ < 1$. The solutions of $f_\varrho(\xi) = \xi$ in $(0, 1]$ are $\xi \in \{\xi_\varrho^-, \xi_\varrho^+\}$. For $\varrho \in (8/9, 1)$, f_ϱ is deflationary on $(0, \xi_\varrho^-)$, stationary at ξ_ϱ^-, inflationary on $(\xi_\varrho^-, \xi_\varrho^+)$, stationary at ξ_ϱ^+, and deflationary on $(\xi_\varrho^+, 1]$.

For any map $h: (0, 1] \to (0, 1]$ and $H \in \{0, 1, 2, \ldots\}$, define $h^{(H)}$: $(0, 1] \to (0, 1]$ by $h^{(0)}(x) = x$ and $h^{(H+1)}(x) = h[h^{(H)}(x)]$.

LEMMA 2.1. For $\varrho \in (8/9, 1]$ and $x, y \in (\xi_\varrho^-, \xi_\varrho^+)$, there exists $F \in \{0, 1, 2, \ldots\}$ such that $f_\varrho^{(F)}(x) \geqslant y$.

Proof. Since f_ϱ is inflationary on $(\xi_\varrho^-, \xi_\varrho^+)$, the sequence $f_\varrho^{(F)}(x)$ is increasing and bounded above by ξ_ϱ^+ for $F \in \{0, 1, 2, \ldots\}$. Let $\xi \in (\xi_\varrho^-, \xi_\varrho^+]$ denote the limit of this sequence. Taking the limit as $F \to \infty$ on both sides of

$$f_\varrho^{(F+1)}(x) = f_\varrho[f_\varrho^{(F)}(x)],$$

and using the continuity of f_ϱ, we see that $\xi = f_\varrho(\xi)$. Since f_ϱ is stationary at ξ, we have $\xi = \xi_\varrho^+$. Since $y < \xi$, we must have $f_\varrho^{(F)}(x) \geqslant y$ for some $F \in \{0, 1, 2, \ldots\}$. □

For $\varrho \in (8/9, 1)$ and $x, y \in (\xi_\varrho^-, \xi_\varrho^+)$, define

$$S_\varrho(x, y) = \min\{F \in \{0, 1, 2, \ldots\} : f_\varrho^{(F)}(x) \geqslant y\}.$$

LEMMA 2.2. For every $\varrho \in (0, 1)$ and $x, y \in (0, 1)$, there exists $G \in \{0, 1, 2, \ldots\}$ such that $g_\varrho^{(G)}(x) \leqslant y$.

Proof. Similar to Lemma 2.1. □

Let $\varrho_\infty = (4 + 3\sqrt{2})/9 = 0.9158\ldots$.

LEMMA 2.3. We have

$$g_\varrho(\xi_\varrho^+) \begin{cases} \leqslant \xi_\varrho^-, & \text{if } \varrho \in [8/9, \varrho_\infty]; \\ > \xi_\varrho^-, & \text{if } \varrho \in (\varrho_\infty, 1]. \end{cases}$$

Proof. The condition $g_\varrho(\xi_\varrho^+) = \xi_\varrho^-$ is by definition equivalent to

$$\left(\frac{3 + \sqrt{9 - 8/\varrho}}{4}\right)^3 = (1/\varrho)\left(\frac{3 - \sqrt{9 - 8/\varrho}}{4}\right).$$

Expanding the cube, segregating the radical, and squaring yields the cubic $2/\varrho^3 + 72/\varrho^2 - 81/\varrho = 0$ with solutions $1/\varrho \in \{-(27\sqrt{2} + 36)/2, 0, (27\sqrt{2} - 36)/2\}$. Thus, ϱ_∞ is the unique value of ϱ in $(0, 1)$ for which $g_\varrho(\xi_\varrho^-) = \xi_\varrho^+$. Since ξ_ϱ^+ and therefore also $g_\varrho(\xi_\varrho^+)$ are increasing in ϱ, while ξ_ϱ^- is decreasing, the Lemma follows. □

For $\varrho \in (0, 1)$ and $x, y \in (0, 1)$, define

$$Q_\varrho(x, y) = \min\{G \in \{0, 1, 2, \ldots\} : g_\varrho^{(G)}(x) \leqslant y\}.$$

The set $\{f,g\}^*$ of finite words over the alphabet $\{f,g\}$ forms a monoid with concatenation (which will be denoted by juxtaposition) as its operation and the empty word (which will be denoted Λ) as its neutral element. For $w \in \{f,g\}^*$ and $x \in [0,1]$, define

$$\Phi_\varrho(w,x) = \begin{cases} x, & \text{if } w = \Lambda; \\ f_\varrho[\Phi_\varrho(v,x)], & \text{if } w = fv; \\ g_\varrho[\Phi_\varrho(v,x)] & \text{if } w = gv. \end{cases}$$

For $w \in \{f,g\}^*$, let $[f:w]$ and $[g:w]$ denote the number of occurrences of f and g, respectively, in w. For $F, G \in \{0,1,2,\ldots\}$ and $x \in [0,1]$, define

$$\Psi_\varrho(F,G,x) = \max_w \Phi_\varrho(w,x),$$

where the maximum is over all $w \in \{f,g\}^*$ such that $[f:w] = F$ and $[g:w] = G$. For $G \in \{0,1,2,\ldots\}$ and $x \in [0,1]$, define

$$\Psi_\varrho(G,x) = \sup_F \Psi_\varrho(F,G,x),$$

where the supremum is over $F \in \{0,1,2,\ldots\}$. For $x \in [0,1]$, define

$$\Psi_\varrho(x) = \inf_G \Psi_\varrho(G,x),$$

where the infimum is over $G \in \{0,1,2,\ldots\}$.

THEOREM 2.4. If $\varrho \in (0,\varrho_\infty]$, then $\Psi_\varrho(x) = 0$ for all $x \in [0,1]$. If $\varrho \in (\varrho_\infty, 1)$, then

$$\Psi_\varrho(x) = \begin{cases} 0, & \text{if } x \in (0,\xi_\varrho^-]; \\ \xi_\varrho^+, & \text{if } x \in (\xi_\varrho^-, 1]. \end{cases}$$

Proof. Suppose $\varrho \in (0,\varrho_\infty]$. Since f_ϱ and g_ϱ are nondecreasing, so is Ψ_ϱ, so it will suffice to show that $\Psi_\varrho(1) = 0$. We observe that $f_\varrho(x)/x$ assumes its maximum, $9\varrho/8$, at $x = 3/4$, so that

$$g_\varrho(x) \leqslant f_\varrho(x) \leqslant (9\varrho/8)x.$$

Thus $\Psi_\varrho(F, G, x) \leqslant (9\varrho/8)^{F+G}x$. If $\varrho \in (0, 8/9)$, then $9\varrho/8 < 1$, and the desired conclusion is immediate.

Suppose then that $\varrho \in [8/9, \varrho_\infty]$. Since f_ϱ is noninflationary on $[\xi_\varrho^+, 1]$,

$$\Psi_\varrho[Q_\varrho(1, \xi_\varrho^+), 1] \leqslant \xi_\varrho^+. \tag{2.1}$$

Since f_ϱ maps $(0, \xi_\varrho^+]$ and $(0, \xi_\varrho^-]$ into themselves, Lemma 2.3 implies

$$\Psi_\varrho(1, \xi_\varrho^+) \leqslant \xi_\varrho^-.$$

Since f_ϱ is noninflationary on $(0, \xi_\varrho^-]$,

$$\Psi_\varrho[Q_\varrho(\xi_\varrho^-, y), \xi_\varrho^-] \leqslant y, \tag{2.2}$$

for any $y \in (0, \xi_\varrho^-]$. Thus,

$$\Psi_\varrho[Q_\varrho(1, \xi_\varrho^+) + 1 + Q_\varrho(\xi_\varrho^-, y), 1] \leqslant y.$$

Letting $y \to 0$ yields the desired conclusion.

Now suppose $\varrho \in (\varrho_\infty, 1)$. For $x \in (0, \xi_\varrho^-]$, it will suffice to show that $\Psi_\varrho(\xi_\varrho^-) = 0$. But this follows from (2.2). Suppose then that $x \in (\xi_\varrho^-, 1)$. In view of (2.1), it will suffice to show that

$$\Psi_\varrho(x) \geqslant y \tag{2.3}$$

for every $y \in (\xi_\varrho^-, \xi_\varrho^+)$.

By Lemma 2.3, $g_\varrho(\xi_\varrho^+) > \xi_\varrho^-$. Take $\eta^- \in (\xi_\varrho^-, g_\varrho(\xi_\varrho^+))$ and $\eta^+ = g_\varrho^{-1}(\eta^-) \in (g_\varrho^{-1}(\xi_\varrho^-), \xi_\varrho^+)$. Then we have

$$\Psi_\varrho[S_\varrho(3/4, \eta^+) + S_\varrho(\eta^-, 3/4), 1, 3/4] \geqslant \Phi_\varrho(w, 3/4) \geqslant 3/4,$$

where

$$w = f^{S_\varrho(3/4, \eta^+)} g f^{S_\varrho(\eta^-, 3/4)}.$$

Thus

$$\Psi_\varrho\{G[S_\varrho(3/4, \eta^+) + S_\varrho(\eta^-, 3/4)], G, 3/4\} \geqslant \Phi_\varrho(w^G, 3/4) \geqslant 3/4.$$

Finally,

$$\Psi_\varrho\{S_\varrho(x,3/4) + G[S_\varrho(3/4,\eta^+) + S_\varrho(\eta^-,3/4)] + S_\varrho(3/4,y), G, x\}$$
$$\geqslant \Phi_\varrho(v,x) \geqslant y,$$

where

$$v = f^{S_\varrho(3/4,y)} w^G f^{S_\varrho(x,3/4)}.$$

Letting $G \to \infty$ proves (2.3). □

3. THE OPTIMAL RATE

Henceforth, let us assume $\varrho \in (\varrho_\infty, 1)$. For $x \in (\xi_\varrho^-, 1]$, $y \in (0, \xi_\varrho^+)$ and $G \in \{0,1,2,\ldots\}$, define

$$T_\varrho(G,x,y) = \min\{F \in \{0,1,2,\ldots\} : \Psi_\varrho(F,G,x) \geqslant y\}.$$

Note that $T_\varrho(0,x,y) = S_\varrho(x,y)$.

LEMMA 3.1. For $\varrho \in (\varrho_\infty, 1)$, $G \in \{0,1,2,\ldots\}$, $x \in (\xi_\varrho^-, 1]$ and $y \in (0, \xi_\varrho^+)$, we have

$$T_\varrho[G, g_\varrho(\xi_\varrho^+), y] \leqslant T_\varrho[G + Q_\varrho(x, \xi_\varrho^+) + 1, x, y]$$

and

$$T_\varrho[G, x, g_\varrho^{-1}(\xi_\varrho^-)] \leqslant T_\varrho[G + Q_\varrho(\xi_\varrho^-, y) + 1, x, y].$$

Proof. Let $w \in \{f,g\}^*$ be such that $[f:w] = T_\varrho[G + Q_\varrho(x,\xi_\varrho^+) + 1, x, y]$, $[g:w] = G + Q_\varrho(x,\xi_\varrho^+) + 1$ and $\Phi_\varrho(w,x) \geqslant y$. Let $w = ugv$, where $[g:v] = Q_\varrho(x,\xi_\varrho^+)$.

Since f_ϱ is noninflationary on $(\xi_\varrho^+, 1]$, we have $\Phi_\varrho(v,x) \leqslant \xi_\varrho^+$ and $\Phi_\varrho(gv,x) \leqslant g_\varrho(\xi_\varrho^+)$. Thus we have $\Phi_\varrho[u, g_\varrho(\xi_\varrho^+)] \geqslant \Phi_\varrho[u, \Phi_\varrho(gv,x)] \geqslant y$. Since $[g:u] = G$, we have $T_\varrho[G, g(\xi_\varrho^+), y] \leqslant [f:u] \leqslant [f:w] = T_\varrho[G + Q_\varrho(x,\xi_\varrho^+) + 1, x, y]$.

Now let $w \in \{f,g\}^*$ be such that $[f:w] = T_\varrho[G + Q_\varrho(\xi_\varrho^-, y) + 1, x, y]$, $[g:w] = G + Q_\varrho(\xi_\varrho^-, y) + 1$ and $\Phi_\varrho(w,x) \geqslant y$. Let $w = ugv$, where $[g:u] = Q_\varrho(\xi_\varrho^-, y)$.

We must have $\Phi_\varrho(v,x) \geqslant g_\varrho^{-1}(\xi_\varrho^-)$, for if $\Phi_\varrho(v,x) < g_\varrho^{-1}(\xi_\varrho^-)$, then $\Phi_\varrho(gv,x) < \xi_\varrho^-$ and, since f_ϱ is deflationary on $(0,\xi_\varrho^-)$, $\Phi_\alpha(ugv,x) < y$,

a contradiction. Since $[g:v] = G$, we have $T_\varrho[G, x, g_\varrho^{-1}(\xi_\varrho^-)] \leqslant [f:u] \leqslant [f:w] = T_\varrho[G + Q_\varrho(\xi_\varrho^-, y) + 1, x, y]$. □

THEOREM 3.2. For every $\varrho \in (\varrho_\infty, 1)$, there exists a real number $\Theta_\varrho \in (0, \infty)$ such that

$$\lim_{G\to\infty} T_\varrho(G, x, y)/G = \Theta_\varrho$$

for all $x \in (\xi_\varrho^-, 1]$ and $y \in (0, \xi_\varrho^+)$.

Proof. Given $G \in \{0, 1, 2, \ldots\}$, let v be a word in $\{f, g\}^*$ such that $[g:v] = G$, $\Phi_\varrho(v, 3/4) \geqslant 3/4$ and $[f:v] = T_\varrho(G, 3/4, 3/4)$. Similarly, given $H \in \{0, 1, 2, \ldots\}$, let w be a word in $\{f, g\}^*$ such that $[g:w] = H$, $\Phi_\varrho(w, 3/4) \geqslant 3/4$, and $[f:w] = T_\varrho(H, 3/4, 3/4)$. Then $[g:vw] = G + H$ and $\Phi_\varrho(vw, 3/4) \geqslant 3/4$, so

$$T_\varrho(G + H, 3/4, 3/4) \leqslant [f:vw] = T_\varrho(G, 3/4, 3/4) + T_\varrho(H, 3/4, 3/4).$$

If a function T is nonnegative and subadditive [that is, if $T(G + H) \leqslant T(G) + T(H)$ for all G, $H \in \{0, 1, 2, \ldots\}$], then $T(G)/G$ tends to a nonnegative limit as $G \to \infty$; this is Fekete's lemma (see Pólya and Szegö [PS], Pt. I, Ch. 3, §1, Pr. 98). Thus the sequence $T_\varrho(G, 3/4, 3/4)/G$ tends to a limit as $G \to \infty$. Let Θ_ϱ denote this limit.

Suppose that $x \in (\xi_\varrho^-, 3/4)$. Then we have

$$T_\varrho(G, 3/4, 3/4) \leqslant T_\varrho(G, x, 3/4) \leqslant S_\varrho(x, 3/4) + T_\varrho(G, 3/4, 3/4).$$

Dividing by G and letting $G \to \infty$ we have

$$\lim T_\varrho(G, x, 3/4)/G = \Theta_\varrho \tag{3.1}$$

Next suppose that $x \in [3/4, \xi_\varrho^+)$. Then we have

$$T_\varrho(G, x, 3/4) \leqslant T_\varrho(G, 3/4, 3/4) \leqslant S_\varrho(3/4, x) + T_\varrho(G, x, 3/4),$$

and again we have (3.1).

Finally suppose that $x \in [\xi_\varrho^+, 1)$. Then by Lemma 2.3, $g_\varrho(\xi_\varrho^+) \in (\xi_\varrho^-, \xi_\varrho^+)$ and by Lemma 3.1, we have

$$\begin{aligned} T_\varrho[G, g_\varrho(\xi_\varrho^+), 3/4] &\leqslant T_\varrho[G + Q_\varrho(x, \xi_\varrho^+) + 1, x, 3/4] \\ &\leqslant T_\varrho[G + Q_\varrho(x, \xi_\varrho^+) + 1, g_\varrho(\xi_\varrho^+), 3/4], \end{aligned}$$

and again we obtain (3.1). Thus the limit of $T_\varrho(G, x, 3/4)/G$ as $G \to \infty$ is independent of x for $x \in (\xi_\varrho^-, 1]$. Similar arguments show that the limit of $T_\varrho(G, x, y)/G$ as $G \to \infty$ is independent of y for $y \in (0, \xi_\varrho^+)$. □

LEMMA 3.3. For every $\varrho \in (\varrho_\infty, 1)$, there exists a unique $\alpha_\varrho \in (0, 1)$ such that

$$f_\varrho[g_\varrho(x)] < g_\varrho[f_\varrho(x)] \qquad \text{if } x \in (0, \alpha_\varrho),$$
$$f_\varrho[g_\varrho(x)] = g_\varrho[f_\varrho(x)] \qquad \text{if } x = \alpha_\varrho$$

and

$$f_\varrho[g_\varrho(x)] > g_\varrho[f_\varrho(x)] \qquad \text{if } x \in (\alpha_\varrho, 1].$$

Proof. Since f_ϱ and g_ϱ are cubic polynomials, $\langle f_\varrho, g_\varrho \rangle = f_\varrho \circ g_\varrho - g_\varrho \circ f_\varrho$ is a nonic polynomial. It has a sextuple root at 0, and thus has three other roots. For $\varrho \in (\varrho_\infty, 1)$, we have

$$\langle f_\varrho, g_\varrho \rangle(\alpha) \sim (3\varrho^3 - 27\varrho^4)\alpha^6 < 0 \qquad \text{as } \alpha \to 0 \text{ with } \alpha > 0. \quad (3.2)$$

Furthermore, we have

$$\langle f_\varrho, g_\varrho \rangle(1) = 3(\varrho^3 - \varrho^4) > 0, \quad (3.3)$$
$$\langle f_\varrho, g_\varrho \rangle(3) = 3^7(\varrho^3 - 27\varrho^4) < 0$$

and

$$\langle f_\varrho, g_\varrho \rangle(4) = 3 \cdot 2^{12}(\varrho^3 - \varrho^4) > 0.$$

It follows that each of the intervals $(0, 1)$, $(1, 3)$, and $(3, 4)$ contains one root of $\langle f_\varrho, g_\varrho \rangle$. Thus there is a unique $\alpha_\varrho \in (0, 1)$ such that $f_\varrho[g_\varrho(\alpha_\varrho)] = g_\varrho[f_\varrho(\alpha_\varrho)]$. Since $\langle f_\varrho, g_\varrho \rangle$ cannot change sign in the intervals $(0, \alpha_\varrho)$ and $(\alpha_\varrho, 1]$, its signs throughout these intervals are determined by its behavior (3.2) at 0 and its value (3.3) at 1. □

For $\varrho \in (\varrho_\infty, 1)$, define $\beta_\varrho = f_\varrho[g_\varrho(\alpha_\varrho)] = g_\varrho[f_\varrho(\alpha_\varrho)]$, $\eta_\varrho^- = g_\varrho(\alpha_\varrho)$ and $\eta_\varrho^+ = f_\varrho(\alpha_\varrho)$.

LEMMA 3.4. For $\varrho \in (\varrho_\infty, 1)$, we have

$$\xi_\varrho^- < \eta_\varrho^- < \begin{Bmatrix} \alpha_\varrho \\ \beta_\varrho \end{Bmatrix} < \eta_\varrho^+ < \xi_\varrho^+.$$

Proof. To prove the leftmost inequality, let $\xi_\varrho^0 = g_\varrho^{-1}(\xi_\varrho^-)$. By Lemma 2.3, $\xi_\varrho^0 \in (\xi_\varrho^-, \xi_\varrho^+)$. Since f_ϱ is inflationary on $(\xi_\varrho^-, \xi_\varrho^+)$, $f_\varrho(\xi_\varrho^0) > \xi_\varrho^0$. Since g_ϱ is increasing, $g_\varrho[f_\varrho(\xi_\varrho^0)] > g_\varrho(\xi_\varrho^0) = f_\varrho[g_\varrho(\xi_\varrho^0)]$. Thus $\langle f_\varrho, g_\varrho \rangle(\xi_\varrho^0) < 0$ and, by Lemma 3.3, $\xi_\varrho^0 < \alpha_\varrho$. Since g_ϱ is increasing, $\xi_\varrho^- = g_\varrho(\xi_\varrho^0) < g_\varrho(\alpha_\varrho) = \eta_\varrho^-$. This proves the leftmost inequality.

Next we prove the rightmost inequality. By Lemma 2.3, $g_\varrho(\xi_\varrho^+) \in (\xi_\varrho^-, \xi_\varrho^+)$. Since f_ϱ is stationary at ξ_ϱ^+ and inflationary on $(\xi_\varrho^-, \xi_\varrho^+)$, $f_\varrho[g_\varrho(\xi_\varrho^+)] - g_\varrho[f_\varrho(\xi_\varrho^+)] = f_\varrho[g_\varrho(\xi_\varrho^+)] - g_\varrho(\xi_\varrho^+) > 0$. Thus $\langle f_\varrho, g_\varrho \rangle(\xi_\varrho^+) > 0$ and, by Lemma 3.3, $\alpha_\varrho < \xi_\varrho^+$. Since f_ϱ maps $(\xi_\varrho^-, \xi_\varrho^+)$ into itself, $\eta_\varrho^+ = f_\varrho(\alpha_\varrho) < \xi_\varrho^+$. This proves the rightmost inequality.

The remaining inequalities follow from the inflationarity of f_ϱ and the deflationarity of g_ϱ on $(\xi_\varrho^-, \xi_\varrho^+)$. □

Let us fix $\varrho \in (\varrho_\infty, 1)$. Let us say that $w \in \{f, g\}^*$ is *optimal* for $F, G \in \{0, 1, 2, \ldots\}$ and $x \in (\xi_\varrho^-, 1]$ if $[f : w] = F$, $[g : w] = G$ and

$$\Phi_\varrho(w, x) = \Psi_\varrho(F, G, x).$$

Let us order the words $\{f, g\}^H$ of length H from $\{f, g\}^*$ *antilexicographically*, with f preceding g and letters to the right being more significant than letters to the left. Let us say that $w \in \{f, g\}^*$ is *strongly optimal* for F, $G \in \{0, 1, 2, \ldots\}$ and $x \in (\xi_\varrho^-, 1]$ if it is optimal for F, G, and x and if it is antilexicographically last among optimal words for F, G, and x.

LEMMA 3.5. If $F \geqslant 1$ and $G \geqslant 1$, an optimal word for F, G, and $x \in (0, \alpha_\varrho)$ must end with f; an optimal word for F, G, and $x \in (\alpha_\varrho, 1]$ must end with g; and a strongly optimal word for F, G, and α_ϱ must end with g.

Proof. First suppose that $x \in (0, \alpha_\varrho)$. Let w be an optimal word for F, G, and x. Since $F \geqslant 1$, we can write $w = vfg^H$ for some $v \in \{f, g\}^*$ and $H \in \{0, 1, 2, \ldots\}$. We must show that $H = 0$.

If $H \geqslant 1$, then $\Phi_\varrho(g^{H-1}, x) \in (0, \alpha_\varrho)$, since g_ϱ maps the interval $(0, \alpha_\varrho)$ into itself. By Lemma 3.3, $\Phi_\varrho(gfg^{H-1}, x) > \Phi_\varrho(fg^H, x)$. Since f_ϱ and g_ϱ are increasing, $\Phi_\varrho(vgfg^{H-1}, x) > \Phi_\varrho(vfg^H, x)$. This contradicts the definition of w and thus the supposition that $H \geqslant 1$.

Now suppose that $x \in (\alpha_\varrho, 1]$. Since f_ϱ maps the interval $(\alpha_\varrho, 1]$ into itself, a similar argument shows that an optimal word for F, G, and x must end with g.

Finally, if an optimal word w for F, G, and α_ϱ ends with f, w must have the form vf, where v is an optimal word for $F-1$, G, and $f_\varrho(\alpha_\varrho) = \eta_\varrho^+$. Since $\eta_\varrho^+ \in (\alpha_\varrho\ 1]$, the preceding case shows that v must end with g. Thus, v must have the form ug, where u is an optimal word for $F-1$, $G-1$, and $g_\varrho(\eta_\varrho^+) = \beta_\varrho$. By Lemma 3.3,

$$\Phi_\varrho(ufg, \alpha_\varrho) = \Phi_\varrho(w, \alpha_\varrho).$$

Thus there is also an optimal word for F, G, and α_ϱ that ends with g. It follows that a strongly optimal word for F, G, and α_ϱ must end with g. □

For $\varrho \in (\varrho_\infty, 1)$ and $x \in (0, 1]$, define

$$h_\varrho(x) = \begin{cases} f_\varrho(x), & \text{if } x \in (0, \alpha_\varrho); \\ g_\varrho(x), & \text{if } x \in [\alpha_\varrho, 1]. \end{cases}$$

For $H \in \{0, 1, 2, \ldots\}$, define

$$e_\varrho^{(H)}(x) = \begin{cases} f, & \text{if } h_\varrho^{(H)}(x) \in (0, \alpha_\varrho); \\ g, & \text{if } h_\varrho^{(H)}(x) \in [\alpha_\varrho, 1]. \end{cases}$$

Let $w_\varrho(H, x) = e_\varrho^{(H-1)} \cdots e_\varrho^{(0)}$.

LEMMA 3.6. If $F + G \geqslant 1$, a strongly optimal w word for F, G, and $x \in (0, 1]$ must have the form $f^I w_\varrho(H, x)$ or $g^I w_\varrho(H, x)$ for some $I \geqslant 1$ and $H \in \{0, 1, 2, \ldots\}$.

Proof. We proceed by induction on FG. If $FG = 0$, then $F = 0$ or $G = 0$. Then $g^G w_\varrho(0, x)$ or $f^F w_\varrho(0, x)$, respectively, is the unique word w satisfying $[f : w] = F$ and $[g : w] = G$, and thus is strongly optimal for F, G, and x.

If $FG \geqslant 1$, then $F \geqslant 1$ and $G \geqslant 1$, so by Lemma 3.5, w must end with $e_\varrho^{(0)}(x)$. By inductive hypothesis, the strongly optimal word v for $F - [f : e_\varrho^{(0)}(x)]$, $G - [g : e_\varrho^{(0)}(x)]$ and $h_\varrho(x)$ must have the form $f^I w_\varrho[H, h_\varrho(x)]$ or $g^I w_\varrho[H, h_\varrho(x)]$ for some $I \geqslant 1$ and $H \in \{0, 1, 2, \ldots\}$. But then $w = ve_\varrho^{(0)}$ has the form $f^I w_\varrho[H, h_\varrho(x)]e_\varrho^{(0)}(x) = f^I w_\varrho(H + 1, x)$ or $g^I w_\varrho[H, h_\varrho(x)]e_\varrho^{(0)}(x) = g^I w_\varrho(H + 1, x)$. □

Let $F_\varrho(H,x) = [f : w_\varrho(H,x)]$ and $G_\varrho(H,x) = [g : w_\varrho(H,x)]$. Let

$$H_\varrho(G,x) = \min\{H \in \{0,1,2,\ldots\} : G_\varrho(H,x) \geqslant G\}$$

and

$$R_\varrho(G,x) = F_\varrho[H_\varrho(G,x),x].$$

LEMMA 3.7. For $\varrho \in (\varrho_\infty, 1)$ and $x \in [\eta_\varrho^-, \eta_\varrho^+]$,

$$\lim_{G\to\infty} R_\varrho(G,x)/G = \Theta_\varrho.$$

Proof. It will suffice to prove the inequalities

$$T_\varrho(G,x,\eta_\varrho^-) \leqslant R_\varrho(G,x) < T_\varrho(G,x,\eta_\varrho^+)$$

for $G \geqslant 1$, for then the Lemma will follow upon dividing by G, letting $G \to \infty$ and applying Theorem 3.2.

To prove the left inequality, let $F = T_\varrho(G,x,\eta_\varrho^-)$ and let w be the strongly optimal word for F, G, and x. By Lemma 3.6, w is of the form $f^I w_\varrho(H,x)$ or $g^I w_\varrho(H,x)$ for some $I \geqslant 1$ and $H \in \{0,1,2,\ldots\}$.

First suppose that $w = f^I w_\varrho(H,x)$ for some $I \geqslant 1$ and $H \in \{0,1,2,\ldots\}$. Since $x \in [\eta_\varrho^-, \eta_\varrho^+]$ and h_ϱ maps $[\eta_\varrho^-, \eta_\varrho^+]$ into itself, $\Phi_\varrho[w_\varrho(H,w),x] \geqslant \eta_\varrho^-$. Since $[g : w_\varrho(H,x)] = G$, we have $T_\varrho(G,x,\eta_\varrho^-) \leqslant [f : w_\varrho(H,x)] = F - I < F$, a contradiction.

Thus we may suppose that $w = g^I w_\varrho(H,x)$ for some $I \geqslant 1$ and $H \in \{0,1,2,\ldots\}$. Since $[g : w_\varrho(H,x)] < G$, we must have $R_\varrho(G,x) \geqslant [f : w_\varrho(H,x)] = F = T_\varrho(G,x,\eta_\varrho^-)$. This proves the left inequality.

To prove the right inequality, let $F = T_\varrho(G,x,\eta_\varrho^+)$ and let w be the strongly optimal word for F, G, and x. By Lemma 3.6, w is of the form $f^I w_\varrho(H,x)$ or $g^I w_\varrho(H,x)$ for some $I \geqslant 1$ and $H \in \{0,1,2,\ldots\}$.

First suppose that $w = g^I w_\varrho(H,x)$ for some $I \geqslant 1$ and $H \in \{0,1,2,\ldots\}$. Since h_ϱ maps $[\eta_\varrho^-, \eta_\varrho^+]$ into itself, $\Phi_\varrho[w_\varrho(H,x),x] \leqslant \eta_\varrho^+$. Since g_ϱ is deflationary, $\Phi_\varrho[g^I w_\varrho(H,x),x] < \eta_\varrho^+$, a contradiction.

Thus we may suppose that $w = f^I w_\varrho(H,x)$ for some $I \geqslant 1$ and $H \in \{0,1,2,\ldots\}$. Since $[g : w_\varrho(H,x)] = G$, $R_\varrho(G,x) \leqslant [f : w_\varrho(H,x)] = F - I < F = T_\varrho(G,x,\eta_\varrho^+)$. This proves the right inequality. □

Let C_ϱ denote the circle obtained by identifying the endpoints of the interval $[\eta_\varrho^-, \eta_\varrho^+]$. Define $k_\varrho : C_\varrho \to C_\varrho$ by

$$k_\varrho(x) = \begin{cases} f_\varrho(x) & \text{if } x \in [\eta_\varrho^-, \alpha_\varrho); \\ g_\varrho(x), & \text{if } x \in [\alpha_\varrho, \eta_\varrho^+). \end{cases}$$

It is easy to check (using the condition $f_\varrho[g_\varrho(\alpha_\varrho)] = g_\varrho[f_\varrho(\alpha_\varrho)]$) that k_ϱ is a homeomorphism of C_ϱ to itself (that is, k_ϱ is a continuous bijection whose inverse is also continuous). Since neither f_ϱ nor g_ϱ is stationary at any point of $[\eta_\varrho^-, \eta_\varrho^+]$, k_ϱ has no fixed points. Since f_ϱ and g_ϱ are increasing, k_ϱ is orientation preserving [that is, as x runs around C_ϱ in a certain sense, $k_\varrho(x)$ runs around C_ϱ in the same sense].

We shall associate with k_ϱ a number $\vartheta(k_\varrho) \in (0, 1)$ called the *rotation number* of k_ϱ. (This definition depends on some assertions that we shall present without proof. For an account including proofs, see Nitecki [Ni], Chap. 1, or Devaney [D], Pt. I, Chap. 14.)

Define the *canonical projection* $\pi_\varrho : \mathbf{R} \to C_\varrho$ by

$$\pi_\varrho(x) = \eta_\varrho^- + (\eta_\varrho^+ - \eta_\varrho^-)(x - \lfloor x \rfloor),$$

so that π_ϱ is continuous and periodic with period 1. We shall say that homeomorphism $\bar{k}_\varrho$ of $\mathbf{R}$ to itself is a *lift* of k_ϱ if $\pi_\varrho \circ \bar{k}_\varrho = k_\varrho \circ \pi_\varrho$. A lift $\bar{k}_\varrho$ of k_ϱ exists, and any two such lifts differ by an integer. Since k_ϱ is orientation preserving, any lift $\bar{k}_\varrho$ is increasing. For any lift $\bar{k}_\varrho$, the ratio $\bar{k}_\varrho^{(G)}(x)/G$ tends to a limit as $G \to \infty$, and this limit is independent of x. This limit, denoted $\vartheta(\bar{k}_\varrho)$, is the rotation number of the lift $\bar{k}_\varrho$. The rotation numbers of any two lifts of k_ϱ differ by an integer. Thus, for precisely one lift $\bar{k}_\varrho$ we have $\vartheta(\bar{k}_\varrho) \in [0, 1)$. The rotation number of this lift, denoted $\vartheta(k_\varrho)$, is the rotation number of k_ϱ. Since k_ϱ has no fixed points, $\vartheta(k_\varrho) \in (0, 1)$.

THEOREM 3.8. We have

$$\Theta_\varrho = \begin{cases} \vartheta(k_\varrho)/[1 - \vartheta(k_\varrho)], & \text{if } \alpha_\varrho < \beta_\varrho; \\ 1, & \text{if } \alpha_\varrho = \beta_\varrho; \\ [1 - \vartheta(k_\varrho)]/\vartheta(k_\varrho), & \text{if } \alpha_\varrho > \beta_\varrho. \end{cases}$$

Proof. Any orbit $x, k_\varrho(x), \ldots, k_\varrho^{(H)}(x), \ldots$ of k_ϱ lifts to an orbit $\bar{x} < \bar{k}_\varrho(\bar{x}) < \cdots < \bar{k}_\varrho^{(H)}(\bar{x}) < \cdots$ of $\bar{k}_\varrho$. By a *cycle* in an orbit of $\bar{k}_\varrho$, we shall mean that portion of such an orbit falling in a half-open interval of length 1. By a *cycle* in an orbit of k_ϱ, we shall mean the image under π_ϱ of a cycle in the lift of such an orbit.

In the definition of k_ϱ, the first case is inflationary and the second case is deflationary. Thus each case must occur at least once during each cycle. In particular, $\vartheta(k_\varrho) \leqslant 1/2$.

If $\alpha_\varrho < \beta_\varrho$, then $f_\varrho([\eta_\varrho^-, \alpha_\varrho)) = [\beta_\varrho, \eta_\varrho^+) \subseteq [\alpha_\varrho, \eta_\varrho^+)$. Thus, the first case in the definition of k_ϱ occurs at most once during each cycle. Thus, for any $x \in [\eta_\varrho^-, \eta_\varrho^+]$,

$$\lim_{H\to\infty} F_\varrho(H, x)/H = \vartheta(k_\varrho).$$

Complementarily,

$$\lim_{H\to\infty} G_\varrho(H, x)/H = 1 - \vartheta(k_\varrho).$$

Reciprocally,

$$\lim_{G\to\infty} H_\varrho(G, x) = 1/[1 - \vartheta(k_\varrho)].$$

Thus, by Lemma 3.7,

$$\begin{aligned}
\Theta_\varrho &= \lim_{G\to\infty} \frac{R_\varrho(G, x)}{G} \\
&= \lim_{G\to\infty} \frac{F_\varrho[H_\varrho(G, x), x]}{H_\varrho(G, x)} \frac{H_\varrho(G, x)}{G} \\
&= \left\{\lim_{G\to\infty} \frac{F_\varrho[H_\varrho(G, x), x]}{H_\varrho(G, x)}\right\}\left[\lim_{G\to\infty} \frac{H_\varrho(G, x)}{G}\right] \\
&= \vartheta(k_\varrho)/[1 - \vartheta(k_\varrho)].
\end{aligned}$$

If $\alpha_\varrho = \beta_\varrho$, then $f_\varrho([\eta_\varrho^-, \alpha_\varrho)) = [\beta_\varrho, \eta_\varrho^+) = [\alpha_\varrho, \eta_\varrho^+)$ and $g_\varrho([\alpha_\varrho, \eta_\varrho^+)) = [\eta_\varrho^-, \beta_\varrho) = [\eta_\varrho^-, \alpha_\varrho)$. Thus, the two cases in the definition of k_ϱ alternate during each cycle, and $\Theta_\varrho = 1$.

Finally, If $\alpha_\varrho > \beta_\varrho$, then $g_\varrho([\alpha_\varrho, \eta_\varrho^+)) = [\eta_\varrho^-, \beta_\varrho) \subseteq [\eta_\varrho^-, \alpha_\varrho)$. Thus, the second case in the definition of k_ϱ occurs at most once during each cycle. Arguments similar to those for $\alpha_\varrho < \beta_\varrho$ complete the proof. □

4. ASYMPTOTIC PROPERTIES OF THE OPTIMAL RATE

LEMMA 4.1. Suppose that $w_\varrho(H + 2, x) = fg^H f$ for some $\varrho \in (\varrho_\infty, 1)$, $H \in \{0, 1, 2, \ldots\}$ and $x \in [\eta_\varrho^-, \eta_\varrho^+]$. Then

$$Q_\varrho(\beta_\varrho, \alpha_\varrho) \leqslant H \leqslant Q_\varrho(\eta_\varrho^+, \eta_\varrho^-).$$

Proof. Since $w_\varrho(H+2,x)$ ends with f, we have $x \in [\eta_\varrho^-, \alpha_\varrho)$. Thus $\Phi_\varrho(f,x) \in f_\varrho([\eta_\varrho^-, \alpha_\varrho)) = [\beta_\varrho, \eta_\varrho^+)$. Similarly, since $w_\varrho[1, \Phi_\varrho(g^H f, x)]$ ends with f, we have $\Phi_\varrho(g^H f, x) \in [\eta_\varrho^-, \alpha_\varrho)$. Thus $g_\varrho^{(H)}$ sends some element of $[\beta_\varrho, \eta_\varrho^+)$ into some element of $[\eta_\varrho^-, \alpha_\varrho)$. □

LEMMA 4.2. Suppose that $w_\varrho(H+2,x) = g f^H g$ for some $\varrho \in (\varrho_\infty, 1)$, $H \in \{0,1,2,\ldots\}$ and $x \in [\eta_\varrho^-, \eta_\varrho^+]$. Then

$$S_\varrho(\beta_\varrho, \alpha_\varrho) \leqslant H \leqslant S_\varrho(\eta_\varrho^-, \eta_\varrho^+).$$

Proof. Analogous to that for Lemma 4.1. □

THEOREM 4.3. We have

$$\Theta_\varrho \sim \frac{2}{\log_3[1/(1-\varrho)]} \qquad \text{as } \varrho \to 1$$

and

$$\Theta_\varrho \sim \left(\log \frac{3+\sqrt{2}}{2}\right)\left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \left(\log \frac{4-\sqrt{2}}{2}\right)\left(\log \frac{2+\sqrt{2}}{2}\right)$$

$$\text{as } \varrho \to \varrho_\infty.$$

Proof. We shall deal first with the case $\varrho \to 1$. By Lemma 4.1, there are at least $Q_\varrho(\beta_\varrho, \alpha_\varrho)$ and at most $Q_\varrho(\eta_\varrho^+, \eta_\varrho^-)$ occurrences of g between any successive pair of occurrences of f in the word $w_\varrho(H,x)$, for any $H \in \{0,1,2,\ldots\}$ and any $x \in [\eta_\varrho^-, \eta_\varrho^+]$. We shall prove

$$Q_\varrho(\beta_\varrho, \alpha_\varrho) \sim \tfrac{1}{2} \log_3 \frac{1}{1-\varrho} \tag{4.1}$$

and

$$Q_\varrho(\eta_\varrho^+, \eta_\varrho^-) \sim \tfrac{1}{2} \log_3 \frac{1}{1-\varrho} \tag{4.2}$$

as $\varrho \to 1$. By Lemma 3.7, this will complete the proof for the case $\varrho \to 1$.

Since $\varrho^{1/2} g_\varrho(x) = (\varrho^{1/2} x)^3$, we have

$$Q_\varrho(x, y) = \left\lceil \log_3 \left(\ln\left(\frac{1}{\varrho^{1/2} y}\right) \Big/ \ln\left(\frac{1}{\varrho^{1/2} x}\right) \right) \right\rceil. \tag{4.3}$$

From the defining condition $f_\varrho[g_\varrho(\alpha_\varrho)] = g_\varrho[f_\varrho(\alpha_\varrho)]$ we see that α_ϱ is the root of $2\alpha^3 - 12\alpha^2 + 18\alpha - (9 - 1/\varrho) = 0$ in the interval $(0, 1)$. From this it follows that

$$\alpha_\varrho = 1 - (1/6)^{1/2}(1 - \varrho)^{1/2} + O(1 - \varrho).$$

From this we obtain

$$\eta_\varrho^- = 1 - (3/2)^{1/2}(1 - \varrho)^{1/2} + O(1 - \varrho),$$
$$\eta_\varrho^+ = 1 - (3/2)(1 - \varrho) + O[(1 - \varrho)^{3/2}]$$

and

$$\beta_\varrho = 1 - (11/2)(1 - \varrho) + O[(1 - \varrho)^{3/2}].$$

Substituting these expansions into (4.3) yields (4.1) and (4.2).

We next deal with the case $\varrho \to \varrho_\infty$. By an argument similar to that for the preceding case, but using Lemma 4.2, it will suffice to prove

$$S_\varrho(\beta_\varrho, \alpha_\varrho) \sim \left(\log \frac{3 + \sqrt{2}}{2}\right)\left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \left(\log \frac{4 - \sqrt{2}}{2}\right)\left(\log \frac{2 + \sqrt{2}}{2}\right) \tag{4.4}$$

and

$$S_\varrho(\eta_\varrho^-, \eta_\varrho^+) \sim \left(\log \frac{3 + \sqrt{2}}{2}\right)\left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \left(\log \frac{4 - \sqrt{2}}{2}\right)\left(\log \frac{2 + \sqrt{2}}{2}\right) \tag{4.5}$$

as $\varrho \to \varrho_\infty$. We do not have a convenient formula like (4.3) for $S_\varrho(x, y)$, so we shall develop bounds.

Let $\bar{f}_\varrho$ be the piecewise-linear function whose graph is formed by the tangents to the graph of f_ϱ at ξ_ϱ^- and ξ_ϱ^+: for $x \in [\xi_\varrho^-, \xi_\varrho^+]$,

$$\bar{f}_\varrho(x) = \begin{cases} \xi_\varrho^- + (x - \xi_\varrho^-) f_\varrho'(\xi_\varrho^-), & \text{if } x \in [\xi_\varrho^-, \zeta_\varrho]; \\ \xi_\varrho^+ + (x - \xi_\varrho^+) f_\varrho'(\xi_\varrho^+), & \text{if } x \in [\zeta_\varrho, \xi_\varrho^+]; \end{cases}$$

where

$$\zeta_\varrho = \frac{\xi_\varrho^+[1 - f_\varrho'(\xi_\varrho^+)] - \xi_\varrho^-[1 - f_\varrho'(\xi_\varrho^-)]}{f_\varrho'(\xi_\varrho^-) - f_\varrho'(\xi_\varrho^+)}$$

is the abscissa of the intersection of the two tangents. Since $f_\varrho''(x) \leqslant 0$ for $x \in [1/2, 1]$, f_ϱ is concave in this interval, and thus its graph lies below the graphs of its tangents at $\xi_\varrho^-, \xi_\varrho^+ \in [1/2, 1]$. Thus $\bar{f}_\varrho(x) \geqslant f_\varrho(x)$ for $x \in [\xi_\varrho^-, \xi_\varrho^+]$. If we define

$$\underline{S}_\varrho(x, y) = \min\{F \in \{0, 1, 2, \ldots\} : \bar{f}_\varrho^{(F)}(x) \geqslant y\},$$

then $\underline{S}_\varrho(x, y) \leqslant S_\varrho(x, y)$. If $x \in [\xi_\varrho^-, \zeta_\varrho]$, then $\bar{f}_\varrho(x) - \xi_\varrho^- = (x - \xi_\varrho^-) f_\varrho'(\xi_\varrho^-)$. Thus

$$\underline{S}_\varrho(x, \zeta_\varrho) = \left\lceil \ln\left(\frac{\zeta_\varrho - \xi_\varrho^-}{x - \xi_\varrho^-}\right) \Big/ \ln f_\varrho'(\xi_\varrho^-) \right\rceil.$$

Similarly, if $y \in (\zeta_\varrho, \xi_\varrho^+)$, then

$$\underline{S}_\varrho(\zeta_\varrho, y) = \left\lceil \ln\left(\frac{\xi_\varrho^+ - \zeta_\varrho}{\xi_\varrho^+ - y}\right) \Big/ \ln [1/f_\varrho'(\xi_\varrho^+)] \right\rceil.$$

If $x \in (\xi_\varrho^-, \zeta_\varrho)$ and $y \in (\zeta_\varrho, \xi_\varrho^+)$, then

$$\underline{S}_\varrho(x, y) \geqslant \underline{S}_\varrho(x, \zeta_\varrho) + \underline{S}_\varrho(\zeta_\varrho, y) - 1. \tag{4.6}$$

Consider the asymptotic behavior of $\underline{S}_\varrho(\zeta_\varrho, \alpha_\varrho)$ as $\varrho \to \varrho_\infty$. Straightforward estimates yield

$$\xi_\varrho^+ - \zeta_\varrho = \frac{15\sqrt{2} - 21}{2} + O(\varrho - \varrho_\infty).$$

Slightly more delicate estimates yield

$$\xi_\varrho^+ - \alpha_\varrho = (27 - 18\sqrt{2})(\varrho - \varrho_\infty) + O[(\varrho - \varrho_\infty)^2].$$

Again, straightforward estimates yield

$$1/f_\varrho'(\xi_\varrho^+) = \frac{2 + \sqrt{2}}{2} + O(\varrho - \varrho_\infty).$$

Combining these estimates yields

$$\underline{S}_\varrho(\zeta_\varrho, \alpha_\varrho) \sim \left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \left(\log \frac{2+\sqrt{2}}{2}\right).$$

Similar arguments yield

$$\underline{S}_\varrho(\zeta_\varrho, \eta_\varrho^+) \sim \left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \left(\log \frac{2+\sqrt{2}}{2}\right),$$

$$\underline{S}_\varrho(\beta_\varrho, \zeta_\varrho) \sim \left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \left(\log \frac{4-\sqrt{2}}{2}\right)$$

and

$$\underline{S}_\varrho(\eta_\varrho^-, \zeta_\varrho) \sim \left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \left(\log \frac{4-\sqrt{2}}{2}\right).$$

Substituting these expansions into (4.6) and using the identity

$$\frac{1}{\left(\log \frac{4-\sqrt{2}}{2}\right)} + \frac{1}{\left(\log \frac{2+\sqrt{2}}{2}\right)} = \frac{\left(\log \frac{3+\sqrt{2}}{2}\right)}{\left(\log \frac{4-\sqrt{2}}{2}\right)\left(\log \frac{2+\sqrt{2}}{2}\right)}$$

yields

$$S_\varrho(\beta_\varrho, \alpha_\varrho) \geqslant \left(\log \frac{3+\sqrt{2}}{2}\right)\left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \times \left(\log \frac{4-\sqrt{2}}{2}\right)\left(\log \frac{2+\sqrt{2}}{2}\right) \tag{4.7}$$

and

$$S_\varrho(\eta_\varrho^-, \eta_\varrho^+) \geqslant \left(\log \frac{3+\sqrt{2}}{2}\right)\left(\log \frac{1}{\varrho - \varrho_\infty}\right) \Big/ \times \left(\log \frac{4-\sqrt{2}}{2}\right)\left(\log \frac{2+\sqrt{2}}{2}\right). \tag{4.8}$$

To obtain corresponding upper bounds is a bit more complicated. For $\varepsilon \in [0, 1]$ and $x \in [0, 1 - \varepsilon]$, define

$$e_\varepsilon(x) = x + \varepsilon.$$

If ε is sufficiently small, the graph of f_{ϱ_∞} will intersect the graph of e_ε at two points, say $\zeta^-_{\varrho_\infty,\varepsilon}$ and $\zeta^+_{\varrho_\infty,\varepsilon}$, in the interval $(1/2, 1)$, and we have

$$\zeta^-_{\varrho_\infty,\varepsilon} \to \xi^-_{\varrho_\infty} \qquad \text{and} \qquad \zeta^+_{\varrho_\infty,\varepsilon} \to \xi^+_{\varrho_\infty},$$

as $\varepsilon \to 0$. If $\varrho - \varrho_\infty$ is sufficiently small, then the graph of f_ϱ also intersects the graph e_ε at two points, say $\zeta^-_{\varrho,\varepsilon}$ and $\zeta^+_{\varrho,\varepsilon}$, in the interval $(1/2, 1)$, and we have

$$\zeta^-_{\varrho,\varepsilon} \to \zeta^-_{\varrho_\infty,\varepsilon} \qquad \text{and} \qquad \zeta^+_{\varrho,\varepsilon} \to \zeta^+_{\varrho_\infty,\varepsilon},$$

as $\varrho \to \varrho_\infty$.

Let $\underline{f}_{\varrho,\varepsilon}$ be the piecewise-linear function whose graph is formed by the chords of the graph of f_ϱ from ξ^-_ϱ to $\zeta^-_{\varrho,\varepsilon}$, from $\zeta^-_{\varrho,\varepsilon}$ to $\zeta^+_{\varrho,\varepsilon}$ and from $\zeta^+_{\varrho,\varepsilon}$ to ξ^+_ϱ: for $x \in [\xi^-_\varrho, \xi^+_\varrho]$:

$$\underline{f}_{\varrho,\varepsilon}(x) = \begin{cases} \xi^-_\varrho + (x - \xi^-_\varrho)\left[\dfrac{f_\varrho(\zeta^-_{\varrho,\varepsilon}) - \xi^-_\varrho}{\zeta^-_{\varrho,\varepsilon} - \xi^-_\varrho}\right], & \text{if } x \in [\xi^-_\varrho, \zeta^-_{\varrho,\varepsilon}]; \\ x + \varepsilon, & \text{if } x \in [\zeta^-_{\varrho,\varepsilon}, \zeta^+_{\varrho,\varepsilon}]; \\ \xi^+_\varrho + (x - \xi^+_\varrho)\left[\dfrac{f_\varrho(\zeta^+_{\varrho,\varepsilon}) - \xi^+_\varrho}{\zeta^+_{\varrho,\varepsilon} - \xi^+_\varrho}\right], & \text{if } x \in [\zeta^+_{\varrho,\varepsilon}, \xi^+_\varrho]. \end{cases}$$

Since f_ϱ is concave in $[1/2, 1]$, its graph lies above that of its chords. Thus, $\underline{f}_{\varrho,\varepsilon}(x) \leqslant f_\varrho(x)$ for $x \in [\xi^-_\varrho, \xi^+_\varrho]$. If we define

$$\bar{S}_{\varrho,\varepsilon}(x, y) = \min\{F \in \{0, 1, 2, \ldots\} : \underline{f}^{(F)}_{\varrho,\varepsilon}(x) \geqslant y\},$$

then $\bar{S}_{\varrho,\varepsilon}(x, y) \geqslant S_\varrho(x, y)$. If $x \in (\xi^-_\varrho, \zeta^-_{\varrho,\varepsilon}]$, then

$$\bar{S}_{\varrho,\varepsilon}(x, \zeta^-_{\varrho,\varepsilon}) = \left\lceil \ln\left(\frac{\zeta^-_{\varrho,\varepsilon} - \xi^-_\varrho}{x - \xi^-_\varrho}\right) \Big/ \ln\left(\frac{f_\varrho(\zeta^-_{\varrho,\varepsilon}) - \xi^-_\varrho}{\zeta^-_{\varrho,\varepsilon} - \xi^-_\varrho}\right) \right\rceil. \qquad (4.9)$$

Similarly, if $y \in [\zeta^+_{\varrho,\varepsilon}, \xi^+_\varrho)$, then

$$\bar{S}_{\varrho,\varepsilon}(\zeta^+_{\varrho,\varepsilon}, y) = \left\lceil \ln\left(\frac{\xi^+_\varrho - \zeta^+_{\varrho,\varepsilon}}{\xi^+_\varrho - x}\right) \Big/ \ln\left(\frac{\xi^+_\varrho - \zeta^+_{\varrho,\varepsilon}}{\xi^+_\varrho - f_\varrho(\zeta^+_{\varrho,\varepsilon})}\right) \right\rceil. \quad (4.10)$$

Furthermore,

$$\bar{S}_{\varrho,\varepsilon}(\zeta^-_{\varrho,\varepsilon}, \zeta^+_{\varrho,\varepsilon}) = \left\lceil \frac{\zeta^+_{\varrho,\varepsilon} - \zeta^-_{\varrho,\varepsilon}}{\varepsilon} \right\rceil. \quad (4.11)$$

If $x \in (\xi^-_\varrho, \zeta^-_{\varrho,\varepsilon}]$ and $y \in [\zeta^+_{\varrho,\varepsilon}, \xi^+_\varrho)$, then

$$\bar{S}_{\varrho,\varepsilon}(x, y) \leqslant \bar{S}_{\varrho,\varepsilon}(x, \zeta^-_{\varrho,\varepsilon}) + \bar{S}_{\varrho,\varepsilon}(\zeta^-_{\varrho,\varepsilon}, \zeta^+_{\varrho,\varepsilon}) + \bar{S}_{\varrho,\varepsilon}(\zeta^+_{\varrho,\varepsilon}, y). \quad (4.12)$$

Straightforward asymptotic estimates for the expressions appearing in (4.9) and (4.10) yield

$$\bar{S}_{\varrho,\varepsilon}(\zeta^+_{\varrho,\varepsilon}, \alpha_\varrho) \sim \ln\left(\frac{1}{\varrho - \varrho_\infty}\right) \Big/ \ln\left(\frac{\xi^+_{\varrho_\infty} - \zeta^+_{\varrho_\infty,\varepsilon}}{\xi^+_{\varrho_\infty} - f_{\varrho_\infty}(\zeta^+_{\varrho_\infty,\varepsilon})}\right)$$

and

$$\bar{S}_{\varrho,\varepsilon}(\beta_\varrho, \zeta^-_{\varrho,\varepsilon}) \sim \ln\left(\frac{1}{\varrho - \varrho_\infty}\right) \Big/ \ln\left(\frac{\xi^-_{\varrho_\infty} - f_{\varrho_\infty}(\zeta^-_{\varrho_\infty,\varepsilon})}{\xi^-_{\varrho_\infty} - \zeta^-_{\varrho_\infty,\varepsilon}}\right).$$

Substituting these relations and (4.11) into (4.12) and letting $\varepsilon \to 0$, we obtain

$$S_\varrho(\beta_\varrho, \alpha_\varrho) \leqslant \left(\log\frac{3+\sqrt{2}}{2}\right)\left(\log\frac{1}{\varrho - \varrho_\infty}\right) \Big/ \left(\log\frac{4-\sqrt{2}}{2}\right) \times \left(\log\frac{2+\sqrt{2}}{2}\right),$$

since

$$\lim_{\varepsilon \to 0}\left[\frac{\xi^+_{\varrho_\infty} - \zeta^+_{\varrho_\infty,\varepsilon}}{\xi^+_{\varrho_\infty} - f_{\varrho_\infty}(\zeta^+_{\varrho_\infty,\varepsilon})}\right] = 1/f'_{\varrho_\infty}(\xi^+_{\varrho_\infty})$$

and

$$\lim_{\varepsilon \to 0}\left[\frac{\xi^-_{\varrho_\infty} - f_{\varrho_\infty}(\zeta^-_{\varrho_\infty,\varepsilon})}{\xi^-_{\varrho_\infty} - \zeta^-_{\varrho_\infty,\varepsilon}}\right] = f'_{\varrho_\infty}(\xi^-_{\varrho_\infty}).$$

Similar arguments yield

$$S_\varrho(\eta_\varrho^-, \eta_\varrho^+) \leqslant \left(\log\frac{3+\sqrt{2}}{2}\right)\left(\log\frac{1}{\varrho-\varrho_\infty}\right)\Big/\left(\log\frac{4-\sqrt{2}}{2}\right) \times \left(\log\frac{2+\sqrt{2}}{2}\right).$$

Combining these relations with (4.7) and (4.8) yields (4.4) and (4.5). □

5. FURTHER PROPERTIES OF THE OPTIMAL RATE

PROPOSITION 5.1. The optimal rate Θ_ϱ is nonincreasing and continuous in $\varrho \in (\varrho_\infty, 1)$.

Proof. Since $f_\varrho(x)$ and $g_\varrho(x)$ are nondecreasing in ϱ for every $x \in (0, 1]$, so is $\Phi_\varrho(w, x)$ for every $w \in \{f, g\}^*$ and $\Psi_\varrho(F, G, x)$ for every $F, G \in \{0, 1, 2, \ldots\}$. Thus, $T_\varrho(G, x, y)$ is nonincreasing in ϱ for all $G \in \{0, 1, 2, \ldots\}$, $x \in (\xi_\varrho^-, 1]$ and $y \in (0, \xi_\varrho^+)$. Dividing by G and letting $G \to \infty$, we see that Θ_ϱ is nonincreasing in ϱ.

To show that Θ_ϱ is continuous, let C denote the interval $[0, 1]$ with its endpoints identified. The circle C is a metric space under the metric

$$d_C(x, y) = \min\{|x - y|, |1 - x + y|\}$$

for $x, y \in C$. Let $\mathscr{C}$ denote the set of all orientation-preserving fixed-point-free homeomorphisms from C to itself. The set $\mathscr{C}$ is a metric space under the metric

$$d_{\mathscr{C}}(p, q) = \max_{x \in C} d_C[p(x), q(x)]$$

for all $p, q \in \mathscr{C}$.

For $\varrho \in (\varrho_\infty, 1)$, define $l_\varrho : C_\varrho \to C$ by $l_\varrho(x) = (x - \eta_\varrho^-)/(\eta_\varrho^+ - \eta_\varrho^-)$. Then $\varrho \to l_\varrho \circ k_\varrho \circ l_\varrho^{-1}$ is a continuous map from $(\varrho_\infty, 1)$ to $\mathscr{C}$, since f_ϱ, g_ϱ and α_ϱ are all continuous in ϱ.

Since the rotation number $\vartheta : \mathscr{C} \to (0, 1)$ is continuous (see Devaney [D], Pt. I, Cor. 14.7), $\vartheta(l_\varrho \circ k_\varrho \circ l_\varrho^{-1})$ is continuous in ϱ.

Since l_ϱ is a homeomorphism, $\vartheta(k_\varrho) = \vartheta(l_\varrho \circ k_\varrho \circ l_\varrho^{-1})$ is also continuous in ϱ. Thus, by Theorem 3.8, Θ_ϱ is continuous in ϱ. □

LEMMA 5.2. Let F and G be positive integers with $(F, G) = 1$. For $\varrho \in (\varrho_\infty, 1)$, we have $\Theta_\varrho = F/G$ if and only if there exist $\lambda_1, \ldots, \lambda_{F+G}$ such that

$$\eta_\varrho^- \leqslant \lambda_1 < \cdots < \lambda_F < \alpha_\varrho \leqslant \lambda_{F+1} < \cdots < \lambda_{F+G} < \eta_\varrho^+ \tag{5.1}$$

and

$$\begin{aligned} f_\varrho(\lambda_H) &= \lambda_{H+G}, \qquad \text{if } H \in \{1, \ldots, F\}; \\ g_\varrho(\lambda_H) &= \lambda_{H-F}, \qquad \text{if } H \in \{F+1, \ldots, F+G\}. \end{aligned} \tag{5.2}$$

Proof. If (5.1) and (5.2) hold, then $\lambda_1, \ldots, \lambda_{F+G}$ is an orbit of h_ϱ. That $\Theta_\varrho = F/G$ then follows from Lemma 3.7.

Suppose now that $\Theta_\varrho = F/G$. Then $\vartheta(k_\varrho)$ is rational, by Theorem 3.8. This implies that k_ϱ has a periodic point (see Nitecki [Ni], Ch. 1, Prop. 2 or Devaney [D], Pt. I, Prop. 14.8). Let

$$\eta_\varrho^- \leqslant \lambda_1 < \cdots < \lambda_K < \eta_\varrho^+ \tag{5.3}$$

be a minimal orbit of k_ϱ. Let $K = L + M$, where $\lambda_L < \alpha_\varrho \leqslant \lambda_{L+1}$. Since f_ϱ is increasing and inflationary in $[\eta_\varrho^-, \eta_{s\varrho}^+)$, while g_ϱ is increasing and deflationary, we have

$$\begin{aligned} f_\varrho(\lambda_H) &= \lambda_{H+M}, \qquad \text{if } H \in \{1, \ldots, L\}; \\ g_\varrho(\lambda_H) &= \lambda_{H-L}, \qquad \text{if } H \in \{L+1, \ldots, L+M\}. \end{aligned}$$

We have have $(L, M) = 1$, else $\lambda_{(L,M)}, \lambda_{2(L,M)}, \ldots, \lambda_{L+M}$ would be a proper suborbit of (5.3) contradicting the minimality of (5.3). Finally, applying Lemma 3.7 to any of $\lambda_1, \ldots, \lambda_K$ yields $L = F$ and $M = G$, completing the proof of (5.2). □

Let F and G be positive integers with $(F, G) = 1$, and let $\varrho \in (\varrho_\infty, 1)$ be such that $\Theta_\varrho = F/G$ (such a ϱ exists by virtue of the continuity and asymptotics of Θ_ϱ). Let $\lambda_1, \ldots, \lambda_{F+G}$ satisfy (5.1) and (5.2). Then for $H \in \{1, \ldots, F+G\}$, the successor of λ_H in this orbit of k_ϱ is λ_{H+G}, where the subscript addition is modulo $F+G$ in $\{1, \ldots, F+G\}$.

For $H \in \{1, \dots, F+G\}$ and $I \in \{0, \dots, F+G-1\}$, define

$$e_{F,G,H,I} = \begin{cases} f, & \text{if } H + IG \in \{1, \dots, F\} \pmod{F+G}; \\ g, & \text{if } H + IG \in \{F+1, \dots, F+G\} \pmod{F+G}. \end{cases}$$

Let $w_{F,G,H} = e_{F,G,H,F+G-1} \cdots e_{F,G,H,0}$. Then we have $w_\varrho(F+G, \lambda_H) = w_{F,G,H}$.

For $\varrho \in (\varrho_\infty, 1)$ with $\Theta_\varrho = F/G$, $H \in \{1, \dots, F+G\}$ and $x \in (0, 1]$, define

$$p_{\varrho,H}(x) = \Phi_\varrho(w_{F,G,H}, x).$$

Then λ_H is a fixed point of $p_{\varrho,H}$. Conversely, if λ is a fixed point of $p_{\varrho,H}$, and if we define

$$\mu_{H+IG} = \Phi_\varrho(e_{F,G,H,I-1} \cdots e_{F,G,H,0}, \lambda) \tag{5.4}$$

for $I \in \{0, \dots, F+G-1\}$, then we have

$$\begin{aligned} f_\varrho(\mu_H) &= \mu_{H+G}, \qquad \text{if } H \in \{1, \dots, F\}; \\ g_\varrho(\mu_H) &= \mu_{H-F}, \qquad \text{if } H \in \{F+1, \dots, F+G\}. \end{aligned} \tag{5.5}$$

The elements of $\{\mu_1, \dots, \mu_{F+G}\}$ must be distinct, since if two of them were equal, two cyclic shifts of the word $w_{F,G,H}$ would be equal, contradicting $(F, G) = 1$. Since f_ϱ is inflationary, the largest F elements of $\{\mu_1, \dots, \mu_{F+G}\}$ must be the images under f_ϱ of the F smallest. Since g_ϱ is deflationary, the smallest G elements of $\{\mu_1, \dots, \mu_{F+G}\}$ must be the images under g_ϱ of the G largest. Since f_ϱ and g_ϱ are increasing, it follows that

$$\mu_1 < \cdots < \mu_{F+G}.$$

LEMMA 5.3. For all positive integers F and G with $(F, G) = 1$, there exists an algebraic number $\varrho_{F/G} \in (\varrho_\infty, 1)$ such that for all $\varrho \in (\varrho_\infty, 1)$ and $H \in \{1, \dots, F+G\}$, $p_{\varrho,H}$ has a fixed point in $[1/2, 1]$ if and only if $\varrho \geqslant \varrho_{F/G}$.

Proof. Since the words $w_{F,G,H}$ are cyclic shifts of each other for $H \in \{1, \dots, F+G\}$, either none of the polynomials $p_{\varrho,1}, \dots, p_{\varrho,F+G}$ have a fixed point in $[1/2, 1]$ or all of them do.

Since there exists $\varrho \in (\varrho_\infty, 1)$ such that $\Theta_\varrho = F/G$, there exists $\varrho \in (\varrho_\infty, 1)$ such that $p_{\varrho,H}$ has a fixed point in $[1/2, 1]$. Let $\varrho_{F/G}$ denote the infimum of $\varrho \in (\varrho_\infty, 1)$ such that $p_{\varrho,H}$ has a fixed point in $[1/2, 1]$. By continuity of $p_{\varrho,H}(x)$ in ϱ and x, $p_{\varrho_{F/G},H}$ has a fixed point in $[1/2, 1]$. By Lemma 2.3, $p_{\varrho_\infty,H}$ does not have a fixed point in $[1/2, 1]$. Thus $\varrho_{F/G} \in (\varrho_\infty, 1)$. Furthermore, if $\varrho \geqslant \varrho_{F/G}$, then $p_{\varrho,H}$ has a fixed point in $[1/2, 1]$, since $p_{\varrho,H}(x) \geqslant p_{\varrho_{F/G},H}(x)$ for $x \in [1/2, 1]$, but $p_{\varrho,H}(x) < x$ for $x \in \{1/2, 1\}$. It remains to show that $\varrho_{F/G}$ is an algebraic number.

At a fixed point λ of $p_{\varrho_{F/G},H}$, the graph of this polynomial must be tangent to the diagonal. Thus, λ is a double root of the equation $p_{\varrho_{F/G},H}(\xi) = \xi$, and $\varrho_{F/G}$ is a root of the discriminant of $p_{\varrho,H}(\xi) - \xi$. Since this is a polynomial in ϱ and ξ with integer coefficients, $\varrho_{F/G}$ is an algebraic number. □

THEOREM 5.4. For all positive integers F and G with $(F, G) = 1$, there exist algebraic numbers $\varrho^-_{F/G}$ and $\varrho^-_{F/G}$ in $(\varrho_\infty, 1)$, with $\varrho^-_{F/G} < \varrho^+_{F/G}$, such that for all $\varrho \in (\varrho_\infty, 1)$, $\Theta_\varrho = F/G$ if and only if $\varrho \in [\varrho^-_{F/G}, \varrho^+_{F/G}]$.

Proof. From Theorem 4.3 and Proposition 5.1, it follows that the set of $\varrho \in (\varrho_\infty, 1)$ such that $\Theta_\varrho = F/G$ forms an interval of the form $[\varrho^-_{F/G}, \varrho^+_{F/G}]$, with $\varrho^-_{F/G} \leqslant \varrho^+_{F/G}$. It remains to prove that $\varrho^-_{F/G}$ and $\varrho^+_{F/G}$ are algebraic numbers with $\varrho^-_{F/G} < \varrho^+_{F/G}$.

We claim that $\varrho^-_{F/G} = \varrho_{F/G}$. By Lemma 5.3, the polynomial $p_{\varrho_{F/G},1}$ has a fixed point λ. Define $\mu_1, \ldots, \mu_{F+G}$ according to (5.4) with $\varrho = \varrho_{F/G}$. Then we have (5.5). It remains to prove that

$$\mu_F < \alpha_{\varrho_{F/G}} \leqslant \mu_{F+1}.$$

To prove this, we begin by observing that $w_{F,G,F} = vgf$ and $w_{F,G,F+1} = vfg$ for some word $v \in \{f, g\}^*$. Since $\Phi_{\varrho_{F/G}}(v, x)$ is increasing in x, $p_{\varrho_{F/G},F}(x) - p_{\varrho_{F/G},F-1}(x)$ has the same sign as $\langle f_{\varrho_{F/G}}, g_{\varrho_{F/G}} \rangle(x)$ for $x \in (0, 1]$. Since $\alpha_{\varrho_{F/G}}$ is the unique unique root of $\langle f_{\varrho_{F/G}}, g_{\varrho_{F/G}} \rangle(\xi)$ in the interval $(0, 1]$, it is also the unique root of $p_{\varrho_{F/G},F}(\xi) - p_{\varrho_{F/G},F+1}(\xi)$ in this interval. Furthermore, since $\alpha_{\varrho_{F/G}}$ is a simple root of $\langle f_{\varrho_{F/G}}, g_{\varrho_{F/G}} \rangle(\xi)$, and since the derivative of $\Phi_{\varrho_{F/G}}(v, x)$ with respect to x does not vanish for $x \in (0, 1)$, $\alpha_{\varrho_{F/G}}$ is a simple root of $p_{\varrho_{F/G},F}(\xi) - p_{\varrho_{F/G},F+1}(\xi)$.

If $x > \alpha_{\varrho_{F/G}}$, then $p_{\varrho_{F/G},F}(x) < p_{\varrho_{F/G},F+1}(x)$. Furthermore, $p_{\varrho_{F/G},F+1}(x) \leqslant x$, else $p_{\varrho,F+1}$ would have a fixed point for some $\varrho < \varrho_{F/G}$. Thus if $x > \alpha_{\varrho_{F/G}}$, then $p_{\varrho_{F/G},F}(x) < x$, so any fixed point λ

of $p_{\varrho_{F/G},F+1}$ must satisfy $\lambda \leqslant \alpha_{\varrho_{F/G}}$. Similarly, any fixed point λ of $p_{\varrho_{F/G},F+1}$ must satisfy $\lambda \leqslant \alpha_{\varrho_{F/G}}$. Furthermore, $\alpha_{\varrho_{F/G}}$ cannot be a fixed point of either $p_{\varrho_{F/G},F}$ or $p_{\varrho_{F/G},F+1}$. For if it were a fixed point of one, it would also be a fixed point of the other. Since fixed points of $p_{\varrho_{F/G},F}$ and $p_{\varrho_{F/G},F+1}$ are double roots of $p_{\varrho_{F/G},F}(\xi) - \xi$ and $p_{\varrho_{F/G},F+1}(\xi) - \xi$, respectively, $\alpha_{\varrho_{F/G}}$ would thus be a double root of $p_{\varrho_{F/G},F}(\xi) - p_{\varrho_{F/G},F+1}(\xi)$, contradicting what was shown above.

Since μ_F is a fixed point of $p_{\varrho_{F/G},F}$ and μ_{F+1} is a fixed point of $p_{\varrho_{F/G},F+1}$, we have

$$\mu_F < \alpha_{\varrho_{F/G}} < \mu_{F+1},$$

which completes the proof that $\varrho_{F/G} = \varrho^-_{F/G}$.

For all $\varrho \geqslant \varrho^-_{F/G}$, $p_{\varrho,1}$ has a fixed point λ, from which we may define $\mu_1, \ldots, \mu_{F+G}$ according to (5.4) and for which (5.5) holds. Thus the condition

$$\mu_F < \alpha_\varrho < \mu_{F+1} \tag{5.6}$$

must fail for $\varrho = \varrho^+_{F/G}$. By continuity, we must have $\mu_F = \alpha_\varrho$ or $\mu_{F+1} = \alpha_\varrho$ for $\varrho = \varrho^+_{F/G}$. It follows that $\varrho^+_{F/G}$ is a root of the resultant of $p_{\varrho,F}(\xi) - \xi$ and $\langle f_\varrho, g_\varrho \rangle(\xi)$, or of $p_{\varrho,F+1}(\xi) - \xi$ and $\langle f_\varrho, g_\varrho \rangle(\xi)$. Since these are polynomials in ϱ and ξ with integer coefficients, $\varrho^+_{F/G}$ is an algebraic number. Since (5.6) holds for $\varrho = \varrho^-_{F/G}$, we have $\varrho^-_{F/G} < \varrho^+_{F/G}$. □

For every $\varrho \in (\varrho_\infty, 1)$ and every $H \in \{0, 1, 2, \ldots\}$, $k^{(H)}_\varrho(x)$ is a piecewise polynomial function of x, with finitely many breakpoints that are algebraic functions of ϱ. As a function of ϱ and x, it is a bivariate piecewise-polynomial function, with finitely many breaklines that are algebraic curves. Thus is has continuous partial derivatives with respect to ϱ and x, except on finitely many algebraic curves.

LEMMA 5.5. For all $H \geqslant 1$, $\varrho \in (\varrho_\infty, 1)$ and $x \in (\eta^-_\varrho, \eta^+_\varrho)$, we have $dk^{(H)}_\varrho(x)/d\varrho \geqslant 1/2$, except on finitely many algebraic curves.

Proof. We proceed by induction on H. If $H = 1$, then $k^{(H)}_\varrho(x) = k_\varrho(x)$ and we have

$$\frac{dk_\varrho(x)}{d\varrho} = \left\{ \begin{matrix} f_1(x) > f_\varrho(x) \\ g_1(x) > g_\varrho(x) \end{matrix} \right\} = k_\varrho(x) \geqslant \eta^-_\varrho > 1/2.$$

If $H \geqslant 2$, then we have by the chain rule

$$\frac{dk_\varrho^{(H)}(x)}{d\varrho} = \left.\frac{\partial k_\varrho(y)}{\partial \varrho}\right|_{y=k_\varrho^{(H-1)}(x)} + \left.\frac{\partial k_\varrho(y)}{\partial y}\right|_{y=k_\varrho^{(H-1)}(x)} \frac{dk_\varrho^{(H-1)}(x)}{d\varrho}.$$

The first term is at least $1/2$ by the case $H = 1$. In the second term, the first factor is positive since k_ϱ is increasing between breakpoints; the second factor is positive by inductive hypothesis. Thus, the second term is positive and the sum is at least $1/2$. □

THEOREM 5. For every irrational number $t \in (0, \infty)$, there is a unique $\varrho \in (\varrho_\infty, 1)$ such that $\Theta_\varrho = t$.

Proof. As in the proof of Theorem 5.4, the set of $\varrho \in (\varrho_\infty, 1)$ such that $\Theta_\varrho = t$ forms an interval of the form $[\varrho_t^-, \varrho_t^+]$. It remains to prove that $\varrho_t^- = \varrho_t^+$.

Suppose, to obtain a contradiction, that $\varrho_t^- < \varrho_t^+$. Set $\varrho = (\varrho_t^+ + \varrho_t^-)/2$. Then Θ_ϱ is irrational. By Theorem 3.8, $\vartheta(k_\varrho)$ is also irrational. The homeomorphism k_ϱ is differentiable (except at the points η_ϱ^-, α_ϱ and η_ϱ^+) and its derivative is continuous (except at these points) and has bounded variation. It follows that k_ϱ is transitive (that is, every orbit is dense in C_ϱ). (This is proved for a diffeomorphism with its derivative continuous and of bounded variation in Nitecki [Ni], p. 45; scrutiny of the proof given there reveals that it remains valid under the weaker assumptions available here.)

Choose $\varepsilon > 0$ sufficiently small that $\varepsilon < (\varrho_t^+ - \varrho_t^-)/2$ and the intersection

$$\bigcap_{\varrho-\varepsilon<\sigma<\varrho+\varepsilon} [\eta_\sigma^-, \eta_\sigma^+]$$

contains a nonempty open interval (ζ^-, ζ^+). (This can be done, since η_σ^- and η_σ^+ are continuous in σ and $\eta_\varrho^- < \eta_\varrho^+$.)

Set $\zeta = (\zeta^+ + \zeta^-)/2$, and choose $\delta > 0$ sufficiently small that $\delta < (\zeta^+ - \zeta^-)/2$ and $\delta < \varepsilon/2$. We shall prove that there exists $\sigma \in [\varrho - 2\delta, \varrho + 2\delta] \subseteq [\varrho - \varepsilon, \varrho + \varepsilon] \subseteq [\varrho_t^-, \varrho_t^+]$ such that k_σ has a periodic point. This will imply that $\vartheta(k_\sigma)$ is rational (see Nitecki [Ni], Ch. 1, Prop. 2 or Devaney [D], Pt. I, Prop. 14.8). By Theorem 3.8, this will imply that Θ_σ is rational. Since $\sigma \in [\varrho_t^-, \varrho_t^+]$, this will imply that $\Theta_\sigma = \Theta_\varrho$, so that $\Theta_\varrho = t$ is rational, a contradiction.

Since k_ϱ is transitive, the orbit of ζ is dense in C_ϱ. Thus, $k_\varrho^{(H)}(\zeta) = \zeta'$ for some $\zeta' \in (\zeta^-, \zeta^+)$ and $H \geqslant 1$. Suppose firstly that $\zeta' \in [\zeta, \zeta^+) = [\zeta, \zeta + \delta)$. As σ increases from ϱ to $\varrho + 2\delta$, $k_\sigma^{(H)}(\zeta)$ increases from

$k_\varrho^{(H)}(\zeta)$ to

$$k_{\varrho+2\delta}^{(H)}(\zeta) = k_\varrho^{(H)}(\zeta) + \int_\varrho^{\varrho+2\delta} \frac{dk_\tau^{(H)}(\zeta)}{d\tau}\, d\tau,$$

which is at least $k_\varrho^{(H)}(\zeta) + \delta$, by Lemma 5.5. Thus, by continuity, we have $k_\sigma^{(H)}(\zeta) = \zeta$ for some $\sigma \in [\varrho, \varrho + 2\delta]$. If $\zeta' \in (\zeta^-, \zeta]$, a similar argument shows that $k_\sigma^{(H)}(\zeta) = \zeta$ for some $\sigma \in [\varrho - 2\delta, \varrho]$. □

ACKNOWLEDGMENT

The author is indebted to Michael Shub, who warned him to expect the theorems of Section 5.

REFERENCES

[D] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*. Benjamin/Cummings Publishing, 1986.

[DO1] R. L. Dobrushin and S. I. Ortyukov, "Upper bound for the redundancy of self-correcting arrangements of unreliable functional elements," *Prob. Info. Transm.* 13: 203–218 (1977).

[DO2] R. L. Dobrushin and S. I. Ortyukov, "Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements," *Prob. Info. Transm.* 13: 59–65 (1977).

[Ne] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components." In C. E. Shannon and J. McCarthy (eds.), *Automata Studies*, pp. 43–98. Princeton University Press, Princeton, 1956.

[Ni] Z. Nitecki, *Differentiable Dynamics*. M.I.T. Press, Cambridge, 1971.

[P1] N. Pippenger, "On networks of noisy gates," *IEEE Symp. Found. Computer Sci.* 26: 30–38 (1985).

[P2] N. Pippenger, "Reliable computation by formulae in the presence of noise," *IEEE Trans. Info. Theory* 34: 194–197 (1988).

[PS] G. Pólya and G. Szegö, *Problems and Theorems in Analysis*. Springer-Verlag, Berlin, 1972.

DETERMINISTIC SIMULATION OF PROBABILISTIC CONSTANT DEPTH CIRCUITS

Miklos Ajtai and Avi Wigderson

ABSTRACT

We explicitly construct, for every integer n and $\varepsilon \geqslant 0$, a family of functions (pseudorandom bit generators) $f_{n,\varepsilon}$: $\{0,1\}^{n^\varepsilon} \to \{0,1\}^n$ with the following property: for a random seed, the pseudorandom output "looks random" to any polynomial size, constant depth, unbounded fan-in circuit. Moreover, the functions $f_{n,\varepsilon}$ themselves can be computed by uniform polynomial size, constant depth circuits.

Some (interrelated) consequences of this result are given:

1. Deterministic simulation of probabilistic algorithms. The constant depth analogues of the probabilistic complexity classes *RP* and *BPP* are contained in the deterministic

Advances in Computing Research, Volume 5, pages 199–222.

ISBN: 0-89232-896-7

complexity classes $DSPACE(n^\varepsilon)$ and $DTIME(2^{n^\varepsilon})$ for any $\varepsilon \geqslant 0$.

2. Making probabilistic constructions deterministic. Some probabilistic construction of structures that elude explicit constructions can be simulated in the above complexity classes.
3. Approximate counting. The number of satisfying assignments to a (CNF or DNF) formula, if not too small, can be arbitrarily approximated in $DSPACE(n^\varepsilon)$ and $DTIME(2^{n^\varepsilon})$, for any $\varepsilon > 0$.

We also present two results for the special case of depth 2 circuits. They deal, respectively, with finding a satisfying assignment and approximately counting the number of satisfying assignments. For example, 3-CNF formulas with a fixed fraction of satisfying assignments, both tasks can be performed in polynomial time!

1. INTRODUCTION

The relationship between randomized and deterministic computation is a fundamental issue in the theory of computation. The results on this subject fall into the following categories.

1.1. Simulating Randomness by Nonuniformity

Adleman [Ad] showed that any language in RP can be computed by a polynomial size family of circuits. However, the proof is existential, and there is no known way of explicitly constructing these circuits. A similar result, for simulating probabilistic, polynomial size, constant depth circuits by nonuniform deterministic ones is due to Ajtai and Ben-Or [AB].

1.2. Simulating Randomness under an Unproven Assumption

Yao [Ya] has shown that if one way functions exist, then RP is contained in $DTIME(2^{n^\varepsilon})$, for any fixed positive ε. Note that the assumption is extremely strong, as it implies in particular that $P \neq NP \cap coNP$. Similar results are given in [FLS], who study the space complexity of the simulation, and [RT], who consider RNC instead of RP.

1.3. Simulating Randomness by Alternation

Sipser and Gacs [Si] showed that *BPP* is contained in Δ_2^p. Of course, the time or space complexities of languages in this class are unknown. A related result, due to Stockmeyer [St], is that approximate counting is in Δ_3^p.

1.4. Simulating Specific Randomized Algorithms

By a careful analysis of how randomness is used in a specific algorithm, one may be able to replace it by a deterministic construction. Such examples are the parallel algorithms in [Lu, KUW, KW]. Also related are explicit constructions of graphs with special properties, which can be found in [Ma] and [GG].

There were no explicit upper bounds on the deterministic simulation of any nontrivial class of probabilistic algorithms. In fact, there is no such simulation that does less than brute force enumeration of all possibilities for the random inputs.

We prove in this paper that probabilistic, polynomial size, constant depth, unbounded fan-in circuits can be simulated in $DSPACE(n^{\varepsilon})$ [and hence also in $DTIME(2^{n^{\varepsilon}})$], for every fixed positive ε. This is done by generating a small set of pseudorandom binary strings, such that a randomly chosen one of them "looks random" to any polynomial size, constant depth circuit.

It is interesting to note that our "pseudorandom bit generator" is purely combinatorial, in contrast to the number theoretic generators used in cryptography (e.g., [Sh, BM, BBS]).

The proof that our generator "works" requires an intimate understanding of the structure of constant depth circuits. Such an understanding is drawn from the lower bound proof techniques for such circuits [Aj, FSS]. Moreover, these lower bounds are all "probabilistic" (or "nonconstructive"), and an essential part of building the generator is making them explicit. To this end we use the idea of "k-wise independent" random variables (e.g., see [ACGS, Lu, An, KUW]).

In Section 2 we give definitions and state our main theorem. In Section 3 we discuss applications of the main theorem, and in Section 4 we give the proof. In Section 5 we obtain refined results on depth 2 circuits, and discuss their applications.

2. DEFINITIONS AND THE MAIN THEOREM

A *circuit* C is a directed acyclic graph with node labels. The nodes of indegree zero are labeled with input variables, the nodes of outdegree zero are labeled with output variables, and the rest of the nodes are labeled from $\{AND, OR, NOT\}$. We put no bound on fan-in or fan-out.

The *size* of a circuit C, $s(C)$, is the number of nodes in it. The *depth of* C, $d(C)$, is the length of the longest input–output path. We say that C is an (s, d)-*circuit* if $s(C) \leqslant s$ and $d(C) \leqslant d$.

We shall be interested in families of circuits. Let s, d: $N \to N$ be functions. We say that $\{C_n\}, n = 1, 2, \ldots$ is an (s, d)-*family* if for all n, $s(C_n) \leqslant s(n)$, $d(C_n) \leqslant d(n)$. If $s = n^{O(1)}$, $d = O(1)$ then $\{C_n\}$ is a PC family (polynomial size, constant depth).

A family is uniform if there exists a Turing machine that on input n in unary, outputs a description of C_n, using only $O(\log n)$ space [Ru]. We shall mainly deal with one output circuit. Every such circuit C with n inputs computes a function C: $\{0,1\}^n \to \{0,1\}$ in a natural way. Define $p(C) = Pr[C(x) = 1]$, where $x \in \{0,1\}^n$ with uniform probability.

For inputs that are generated pseudorandomly we use the following. Let f: $\{0,1\}^m \to \{0,1\}^n$ be a function. Define $p_f(C) = Pr[C(x) = 1]$, where $x = f(y)$ and $y \in \{0,1\}^m$ with uniform probability.

Two important parameters measure the "goodness" of f as a pseudorandom bit generator for a circuit C. The natural one is $|p(C) - p_f(C)|$. Another parameter,, for which we get better bounds, is how small can $p(C)$ get so that still $p_f(C)$ does not vanish.

We can now state the main theorem. The present form of the Main Theorem is stronger than the one given in [AW], since the order of quantification has been changed.

MAIN THEOREM. Let ε be fixed. Then there exists a family of functions $\{f_n$: $\{0,1\}^{n^\varepsilon} \to \{0,1\}^n\}$, $n = 1, 2, \ldots$, with the following properties:

(*i*) $\{f_n\}$ can be computed by a uniform, PC family of circuits. (So in particular, $\{f_n\}$ can be computed in LOGSPACE.)

(*ii*) Let l, d, u be fixed integers and $\{C_n\}$ be any (n^l, d) family of circuits. Then for every sufficiently large n,

(*a*) $|p(C_n) - p_{f_n}(C_n)| \leqslant n^{-u}$

(*b*) for a fixed $\tau = \tau(l, d, \varepsilon)$, if $p(C_n) \geqslant 2^{-n^\varepsilon}$, then $p_{f_n}(C_n) > 0$.

3. APPLICATIONS

The applications are given not necessarily in order of importance, but rather in order of notational convenience. All the applications are based on the fact that we can get a fairly good approximation of the output behaviour of C_n by "testing" it on only 2^{n^ε} inputs.

The following notation will be used often. Let g: $N \to [0, 1]$ be a function. We say that g is polynomially small if $g(n)^{-1} = n^{O(1)}$. We say that g is subexponentially small if $g(n)^{-1} = o(2^{n^\varepsilon})$, for every fixed $\varepsilon > 0$.

3.1. Approximate Counting

Let the number of satisfying assignments to a (*CNF* or *DNF*) formula F be $\#F$. Computing $\#F$ from F is $\#P$ complete. It is not known whether $\#P$ is in the polynomial hierarchy. An easier problem is approximate counting, which is in this case to find an integer in the range $[(1 + \beta)^{-1} \#F, (1 + \beta)\#F]$. Call this β approximation. For any polynomially small β, β approximation was shown to be in Δ_3^P by Stockmeyer [S]. No explicit deterministic upper bounds were known for approximate counting. Let $p(F) = \#F/2^n$ be the fraction of satisfying assignments of F, where n is the number of variables in F.

COROLLARY 1. Consider formulas F with polynomially small $p(F)$. Then for every fixed ε and every polynomially small β, the β approximation problem for F is in $DSPACE(n^\varepsilon)$ [and hence also in $DTIME(2^{n^\varepsilon})$].

Proof (*sketch*). F is a polynomial size depth 2 circuit. In $DSPACE(n^\varepsilon)$ all the "seeds" y of f_n can be generated, $f_n(y)$ computed and tested on F. The output is $(\sum_{|y|=n^\varepsilon} F[f_n(y)])2^{n-n^\varepsilon}$. By (*ii*), part (*a*) of the main theorem, it is the desired approximation.

3.2. Easy Cases of Satisfiability

If we are just interested in finding a satisfying assignment to a formula F, the result above can be improved. In [VV], Valiant and

Vazirani showed that finding a satisfying assignment in formulas with exactly one such assignment is essentially as hard as the general case. The following result, which complements theirs, says that if the number of satisfying assignments is large enough, then satisfiability becomes easier.

COROLLARY 2. Consider formulas F with subexponentially small $p(F)$. Then for any $\varepsilon > 0$ a satisfying assignment of F can be found in $DSPACE(n^{\varepsilon})$ [and $DTIME(2^{n^{\varepsilon}})$].

This result follows from (*ii*), part (*b*) of the main theorem.

3.3. Making Probabilistic Constructions Deterministic

Following Sipser [Si], we define a probabilistic construction to be a language $L \subseteq \{0,1\}^* \times \{0,1\}^*$ with the property that if $(u,v) \in L$, then $Pr[(u,x) \in L] \geqslant 1/2$, where x is uniformly chosen with $|x| = |v|$. (u usually gives the size of the required object in unary, and then a random object of the right size has the desired properties.) The deterministic construction problem for L is, on input u, to generate v s.t. $(u,v) \in L$.

If $L \in \sum_i^P$, Sipser calls it a $\sum_i^P$ construction. He shows that if L is a $\sum_i^P$ construction, then the deterministic construction problem for L is, is in $\sum_{i+2}^P$. (An analogous statement is true for $\prod_i^P$.)

Note that $L \in \sum_i^P$ or $L \in \prod_i^P$ means that L can be recognized by a uniform family of constant depth (but possibly exponential size) circuits. We say the L is a PC construction if it can be recognized by a uniform PC family of circuits.

COROLLARY 3. If L is a PC construction, then for any $\varepsilon > 0$ the deterministic construction problem for L is in $DSPACE(n^{\varepsilon})$ [and in $DTIME(2^{n^{\varepsilon}})$].

Note that the uniformity is needed for our deterministic machine to generate the circuit recognizing constructions of size $|u|$, where u is the input.

3.4. Deterministic Simulation of Probabilistic Constant Depth Circuits

A probabilistic circuit C is one with "real" input variables, z, and random input variables, x. If $|z| = n$ and $|x| = m[=m(n)]$,

C computes a function $C: \{0,1\}^{n+m} \to \{0,1\}$. Let $p[C(z)] = Pr[C(z,x) = 1]$ when $x \in \{0,1\}^m$ with uniform probability. The idea of recognizing languages by probabilistic circuits is that the behaviour of $p[C(z)]$ will depend on whether z is in the language or not.

We define two families of complexity classes, $PC1(\alpha)$, $PC2(\alpha)$ where PC refers to polynomial size, constant depth, 1 and 2 refers to whether we allow one or two sided errors, and α is the "accuracy" (in general $\alpha: N \to [0,1]$ is a function). A language $L \subseteq \{0,1\}^*$ is in $PC1(\alpha)$ if there exists a uniform PC family $\{C_n\}$ s.t. for every n and every $z \in \{0,1\}^n$ we have

$$z \in L \to p[C(z)] \geqslant \alpha(n) \text{ and } z \notin L \to p[C(z)] = 0.$$

A language $L \subseteq \{0,1\}^*$ is in $PC2(\alpha)$ if there exists a uniform PC family $\{C_n\}$ s.t. for every n and every $z \in \{0,1\}^n$

$$z \in L \to p[C(z)] \geqslant 1/2 + \alpha(n)$$

$$z \notin L \to p[C(z)] \leqslant 1/2 - \alpha(n).$$

COROLLARY 4. For every fixed $\varepsilon > 0$ we have

(*i*) for every subexponentially small function α, $PC1(\alpha) \subseteq DSPACE(n^\varepsilon)$

(*ii*) for every polynomially small function α, $PC2(\alpha) \subseteq DSPACE(n^\varepsilon)$.

4. PROOF OF THE MAIN THEOREM

The key notion is that of approximating a circuit. A given circuit will undergo a series of simplifications, each restricting the inputs, that will change the output behaviour by only tiny amounts.

The proof has two logical parts. In Part I we show how to approximate $n^{1-\varepsilon}$ input bits of a PC circuit by only $O(\log n)$ bits. In Part II we show how to iterate this construction, adding only a constant to the depth and a polynomial to the size.

4.1. Part I: Sketch of the Proof

As we mentioned, the task in this part is to replace $n^{1-\varepsilon}$ random bits by $O(\log n)$ random bits without affecting the output behavior of the circuit by much, and implement this change by a PC circuit.

Suppose that C is a constant depth polynomial size circuit with the set of variables $A = \{x_1, \ldots, x_n\}$. First we show that if W is a random subset of A with $n^{1-\varepsilon}$ elements and we substitute random values for the variables in $A - W$ then with high probability the resulting circuit will actually depend only on t variables where t is a constant. [See Definition 1. "W is (t, τ)-local."]

If we know such a set W then we can define a pseudorandom input f for C in the following way. Let $f|_{A-W}$ and $f|_W$ be independent, $f|_{A_W}$ uniform on the set of all possible assignments on $A - W$ and let $f|_W$ be uniform on all subsets of W with t elements. [See Definitions 2 and 3; "Y is $(|W|, t)$-uniform."] We will show how to generate such an $f|_W$ only from $O(\log n)$ random bits. For such an f we have that $|Pr[C(x) = 1] - Pr[C(f) = 1]| < n^{-u}$ where u is a large constant.

Unfortunately f still depends on the subset W. To pick a random W with $n^{1-\varepsilon}$ elements requires too many random bits. We will show however that a W with the required properties can be generated from only $O(\log n)$ random bits. To make this part of the argument clearer, first we show those combinatorial properties of a random subset W that guarantee that the circuit we get from C by substituting random bits for the variables in $A - W$ depends only on t elements.

If we consider only depth 2 circuits the essential property of W is the following: if a small subset V of A is given $[|V| \leqslant n^{\beta}$, $\beta + (1 - \varepsilon) < 1]$ then with high probability W will intersect it in a set of constant size. (See Definition 4. "Small intersection property.") This property will imply that if a polynomial family of V's are given then still with high probability W will intersect each of them in a set of constant size. This property will make it possible to replace large conjunctions (or disjunctions) by small ones.

For depth d circuits we have to iterate this argument, so W must be included in a sequence of sets $X_0 \supseteq X_1 \supseteq \cdots \supseteq X_d = W$ so that each X_i has the small intersection property in X_{i-1}. (See Definition 5. "d-iterated small intersection property.)

Now we can define the pseudorandom input f from the $n - n^{1-\varepsilon}$ random bits, which specifies its value on $A - W$, and from the $O(\log n)$ random bits, which describes a W with the d-iterated small intersection property and another $O(\log n)$ random bits, which give a $(|W|, t)$ uniform $f|_W$. So altogether from $n - n^{1-\varepsilon} + O(\log n)$ random bits we defined a pseudorandom input f so that $|p(C) - p_f(C)| < n^{-u}$. In Part II we will iterate this argument to decrease the number of necessary random bits to n^{ε}.

We need some notation. Let C be a circuit on n variables $A = \{x_1, \ldots, x_n\}$. For any subset $Y \subseteq A$ and a binary vector v of length $|Y|$, let $C_{Y,v}$ denote the circuit obtained from C by assigning the values v to the variables in Y, in the natural order. If v is chosen uniformly at random then denote by C_Y the random variable $C_{Y,v}$. Let Z be a random variable taking values from $\{0,1\}^A$, then denote $Pr[C(Z) = 1]$ by $p[C(Z)]$.

Proposition 1 asserts that for a random set of all but $n^{1-\varepsilon}$ of the input variables and a random assignment to them, the resulting circuit will depend only on a constant number of inputs (although it has $n^{1-\varepsilon}$ of them).

DEFINITION 1. We say that a circuit C depends on t variables if there exists a subset $T \subseteq A$ of size t s.t. for every assignment to the variables in T, the resulting circuit is constant. We denote the minimum such t by $t(C)$ and some T of this cardinality by $T(C)$.

If t is an integer and $\tau > 0$ the set $W \subseteq A$ is called (t, τ)-local if

$$Pr[t(C_{A-W}) \leqslant t] \geqslant 1 - 2^{-n^{\tau}}.$$

PROPOSITION 1. Let d, l, u be positive integers and $\varepsilon > 0$. Then there exists an integer t so that if n is sufficiently large, C is a (n^l, d) circuit with n input variables A, and $W \subseteq A$ is random with $|W| = n^{1-\varepsilon}$, then $Pr[t(C_{A-W}) > t] \leqslant n^{-u}$ where we take the probability over the product space "choose W, then choose assignment on $A - W$."

Moreover, there exists a $\tau > 0$ depending only on d, l, u, ε so that with probability at least $1 - n^{-u}$ the set W is (t, τ)-local. (Here the probability is over choices of W only.)

The proof of Proposition 1 is an inductive argument on the depth of the circuit, similar in flavor to the lower bound proofs in [Aj], [FSS], [Ha], and [Ya]. In fact, the first part of Proposition 1 appears with a different proof in [Aj]. The inductive step is based on a property of depth 2 circuits, which will be given in Theorem 1 (Section 5).

We do not prove Proposition 1 now since we later (Proposition 2) will prove a stronger version of it, namely we can pick W randomly from a uniformly given polynomial size family of subsets of A. This remark also relates to Corollary 5.

Note that different choices of v may result in different subsets $T(C_{A-W,v})$ of W that the resulting circuit depends on. However, Proposition 1 tells us that the output distribution of C will essentially remain unchanged if instead of assigning random values to the variables in W (not in $W - A$), we use assignments that "look random" only on t subsets of W. This motivates the next definitions and corollary.

DEFINITION 2. A random variable $Z = Z_1, \ldots, Z_m$ with $Z_i \in \{0,1\}$ is said to be (m,t,p)-uniform if for all $1 \leqslant i \leqslant m$ $Pr(Z_i = 1) = p$ and for every t-subset I of $\{1, \ldots, m\}$ the variables $\{Z_i \mid i \in I\}$ are mutually independent. When $p = 0.5$ we say that Z is (m,t)-uniform.

The key fact about (m,t)-uniform sequences is that they can be simply generated from only $t \log m$ random bits by PC circuits, using polynomials over finite fields. The explicit construction will be given later.

DEFINITION 3. Let $W \subseteq A$ and integer t be fixed. A random variable $Y \in \{0,1\}^n$ is called (W,t)-uniform if $Y|_W$ is $(|W|,t)$-uniform, $Y|_{A-W}$ is uniform, and these two restricted random variables are independent.

Corollary 5 below follows easily from the above definitions and Proposition 1.

COROLLARY 5. Let $d, l, u, \varepsilon, t, \tau$ be as in Proposition 1. For each possible W of size $n^{1-\varepsilon}$ let $Y^{(W)} \in \{0,1\}^n$ be an arbitrary (W,t)-uniform random variable. Then

(1) If W is chosen uniformly at random, then we have $|p(C) - p[C(Y^{(W)})]| \leqslant n^{-u}$.

(2) If W is (t,τ)-local, then $|p(C) - p[C(Y^{(W)})]| \leqslant 2^{-n^{\tau}}$.

(3) If W is (t,τ)-local then there is an evaluation w on W so that $Pr(Y^{(W)}|_W = w) > 0$ and $Pr(C_{W,w} = 1) \geqslant p[C(Y^{(W)})] - 2^{-n^{\tau}}$.

Proof. Fix v in $\{0,1\}^{A-W}$, let $C' = C_{A-W,v}$ (on inputs from W), and consider $p(C')$ vs. $p(C', Y^{(W)}|_W)$. In the first we consider a uniform distribution on inputs to C' and in the second only a t-uniform distribution. The circuit reacts identically if $t(C') \leqslant t$ hence

we get errors only if $t(C') > t$. By Proposition 1, for a random W and v, this happens with probability smaller than n^{-u}. (1)

For a (t, τ)-local set W, this happens with probability smaller than $2^{-n^{\tau}}$. (2)

(3) follows from (2) by averaging over the possible assignments w to W in the random variable $Y^{(W)}$ [replace $p[C(Y^{(W)})]$ by $p(C)$].

We are now in the situation where, if given a (t, τ)-local set W, we can replace its $n^{1-\varepsilon}$ random input bits by $O(\log n)$ random bits. Corollary 5 shows that most W will work, but to specify a random set W we need as many as $n^{1-\varepsilon} \log n$ bits. Our next step will be to generate (t, τ)-local sets W pseudorandomly, using only $O(\log n)$ random bits. This is done by extracting from the proof of Proposition 1 only the essential properties of the random variable W that are actually used.

DEFINITION 4. Let X be a random variable whose values are subsets of a set A of size n. We say that X has the small intersection property with parameters α, β, t if for every set $V \subseteq A$ with $|V| \leqslant n^{\beta}$ we have that if $s \leqslant t$, then

$$Pr(|V \cap X| \leqslant s) \geqslant 1 - n^{-(1-\alpha-\beta)s}.$$

DEFINITION 5. The random variable $X \subseteq A$ has the d-iterated small intersection property with parameters α, β, t if there exists a sequence of random variables $X_1, \ldots, X_d$ so that $X = X_d$, $X_{i+1} \subseteq X_i$ and for any possible fixed values $B_1, \ldots, B_i$ of the variables $X_1, \ldots, X_i$ we have that X_{i+1} with the condition $X_1 = B_1, \ldots, X_i = B_i$ has the small intersection property with parameters α, β, t.

PROPOSITION 2. The consequences of Proposition 1 and Corollary 5 hold if we replace a randomly chosen W of size $n^{1-\varepsilon}$ by a random variable W that has the d-iterated small intersection property with parameters $1 - \varepsilon$, $\varepsilon/2$, t

The proof of Proposition 2 is based on the following Lemmas, 1 and 2.

Before stating the lemmas, we state Theorem 1, which is proved in Section 5. Intuitively, it shows that a depth 2 circuit of small bottom fan-in is almost completely determined by a small subset of its input variables.

THEOREM 1. Let C be a depth 2 circuit of bottom fan-in $\leqslant k$ with input variables in A. Then for every $r \geqslant 2^{2k^2}$ there exists a subset $Q \subseteq A$ s.t.

(1) $|Q| \leqslant r^{2k^2}$
(2) $Pr(C_Q \not\equiv 0, 1) \leqslant 2^{-r}$.

Furthermore, if $|A| = n$, the set Q can be found in time $O(n^{k^2})$.

LEMMA 1. Let k, ε be fixed. For any z if n is sufficiently large, C is a depth 2 circuit with bottom fanin at most k; W is a random variable with the $(1 - \varepsilon, \varepsilon/2, z)$ small intersection property and $t \leqslant z$ then with probability at least $1 - n^{-t\varepsilon/2}$ the set W satisfies the following inequality:

$$Pr[t(C_{A-W}) \leqslant t] \geqslant 1 - 2^{-n^{\varepsilon/5k^2}}.$$

Proof of Lemma 1. Apply Theorem 1 with $r = n^{\varepsilon/4k^2}$ to obtain the set Q. By (1) of the theorem, $|Q| \leqslant r^{2k^2} = n^{\varepsilon/2}$, and since W has the small intersection property, $Pr(|Q \cap W| \leqslant t) \geqslant 1 - n^{-\varepsilon t/2}$.

Now fix W so that $|W \cap Q| \leqslant t$. By (2) of the theorem at most $2^{|Q|-r}$ of the assignments to Q do not determine the value of C. Hence, at most $2^{|Q|-r}$ of the assignments to $Q - W$ will have an extension in $Q \cap W$ that does not fix C, and since these are chosen randomly in C_{A-W} and $|Q - W| \geqslant |Q| - t$ we have $Pr[t(C_{(A-W)}) > t] \leqslant 2^{|Q|-r}/(2^{|Q|-t}) = 2^{-r+t} = 2^{-(n^{\varepsilon/4k^2}+t)}$. If n is sufficiently large then $n^{\varepsilon/4k^2} - t \geqslant n^{\varepsilon/4k^2} - z \geqslant n^{\varepsilon/5k^2}$ so the probability is not greater than $2^{-n^{\varepsilon/5k^2}}$.

Using Lemma 1, we can reduce the depth of a PC circuit C (as in [Aj], [FSS]) by applying it to all the bottom depth 2 circuits of C. Once each of these depends only on t inputs, we change it from CNF to DNF or vice vera without blowing the size up by more than 2^t (a constant) and hence reduce the number of alternations (depth) of C by 1. This is the essence of Lemma 2, and since a pseudo-random W is good enough for Lemma 1, it suffices also for Lemma 2.

LEMMA 2. For all d, l, k, $\varepsilon > 0$ there is a t and a $\tau > 0$ so that for any $z \geqslant t$ if n is sufficiently large and C is an (n^l, d) circuit with

bottom fanin at most k, and W has the small intersection property with parameters $1-\varepsilon$, $\varepsilon/2$, z then with probability at least $1-n^{-[(\varepsilon t/2)-l]}$ the set W will satisfy $Pr(C_{A-W}$ is a depth $d-1$ circuit with bottom fan-in $t) \geqslant 1-2^{-n^r}$.

Now to prove Proposition 2, we simply apply Lemma 2 d times to the circuit (again, as in [Aj], [FSS], [Ya], [Ha]). The resulting circuit (with high probability) depends only on a constant number of inputs, which implies the proposition.

Note that in Definition 4 only intersections of cardinality t or less are important. From this it is easy to deduce:

LEMMA 3. If W is an $(n, n^{-\varepsilon}, t)$-uniform random variable, then $X=\{i \mid W(i)=1\}$ has the small intersection property with parameters $1-\varepsilon, \varepsilon/2, t$.

Proof of Lemma 3. Let $V \subseteq A$ with $|V| \leqslant n^{\varepsilon/2}$, $s \leqslant t$. If $|V \cap X| \geqslant s$ then there are distinct $v_1, \ldots, v_s \in V \cap X$. The number of s-tuples $v_1, \ldots, v_s$ is $\binom{|V|}{s} \leqslant \binom{n^{\varepsilon/2}}{s}$, and for any fixed s-tuple $v_1, \ldots, v_s$ we have $Pr(v_1, \ldots, v_s \in X) \leqslant (n^{-\varepsilon})^s$ since $s \leqslant t$ and W is $(n, n^{-\varepsilon}, t)$-uniform. So $Pr(|V \cap X| \geqslant s) \leqslant \binom{n^{\varepsilon/2}}{s}(n^{-\varepsilon})^s \leqslant n^{s(\varepsilon/2)} n^{-s\varepsilon} = n^{-[1-(1-\varepsilon)-\varepsilon/2]s}$.

Again such a random variable can be constructed from $t \log n$ random bits by PC circuits. To get a random variable with the d-iterated small intersection property one can use d independent constructions as above, which require only $dt \log n$ random bits. We will give the construction in detail in Part II.

To summarize the first part of the proof, we have shown how to replace $n^{1-\varepsilon}$ random bits by $O(\log n)$ random bits, thus reducing the number of inputs by roughly $n^{1-\varepsilon}$ without substantially affecting the output probability of the circuit. This was done by first using $O(\log n)$ bits to specify a pseudorandom set W of size $n^{1-\varepsilon}$ of inputs. Then use other $O(\log n)$ bits to create a pseudorandom assignment to variables in W. The remaining $n-n^{1-\varepsilon}$ inputs receive truly random assignments.

4.2. Part II

At this point it is natural to iterate the construction roughly n^{ε} times. However, this presents some difficulties. For example, if we

implement the construction in the first part, the depth of the circuit increases by a constant, and so we cannot repeat that more than a constant number of times. Another problem that bounds the number of iterations is that we must keep the circuit polynomial in the number of remaining inputs, so we have to stop when at least a polynomial fraction remains.

Conceptually performing this iterative process, we obtain a sequence of roughly n^ε pseudorandom subsets of variables, that together with the remaining part (of size roughly n^ε) form a partition of the set of variables. To each pseudorandom subset we assign (independently) a pseudorandom assignment [requiring a total of $O(n^\varepsilon \log n)$ bits], and to the remaining subset assign random variables (only n^ε).

In order to perform this process in constant depth, we shall generate all parts in the partition together with their assignments simultaneously. We first define the partition assignment pair abstractly, as random variables, and then show how they can be generated from n^ε random bits.

DEFINITION 6. Let d, t be integers and $\delta > 0$. For every n and $0 \leqslant \mu \leqslant n$ define $\langle F, P \rangle$ to be a (d, t, δ)-fooling pair of random variables if the following conditions hold:

(1) Each value of F is a 0,1 assignment to the variables in A, $|A| = n$.
(2) Each value of $P = \langle P_0, \ldots, P_\mu \rangle$ is a sequence of subsets of A so that $P_0, \ldots, P_\mu$ form a partition of A.
(3) For all $0 \leqslant i < \mu$ if $A_0, \ldots, A_{i-1}$ are fixed disjoint subsets of A then the random variable P_i with the condition $P_0 = A_0, \ldots, P_{i-1} = A_{i-1}$ has the d-iterated small intersection property with parameters $1 - 2\delta, \delta, t$.
(4) For all $0 \leqslant i \leqslant \mu$ if $A_0, \ldots, A_i$ are fixed subsets of A then with the condition $P_0 = A_0, \ldots, P_i = A_i$ the random variables $F|_{A_0}, \ldots, F|_{A_i}$ are independent.
(5) For all $0 \leqslant i < \mu$ if $A_i \subseteq A$ then $F|_{A_i}$ with condition $P_i = A_i$ is an $(|A_i|, t)$-uniform random variable.
(6) There is a set $A_\mu \subseteq A$ so that $|A_\mu| = n^\delta$, $Pr(P_\mu = A_\mu) \geqslant 1 - 2^{-n^\delta}$, $Pr(A_\mu \subseteq P_\mu) = 1$, and $F|_{A_\mu}$ has a uniform distribution over all possible assignments on A_μ.

The technical properties of the fooling pair guarantee that it fools any PC circuit with the appropriate parameters.

PROPOSITION 3. For all d, l, u, δ there exists a t such that for all sufficiently large n, $\mu < n$ and a (d, t, δ)-fooling pair $\langle F, P \rangle$ we have the following.

(1) For every (n^l, d) circuit C with n inputs, $|p(C) - p[C(F)]| \leqslant n^{-u}$.
(2) There exists a $\tau > 0$ depending only on d, l, u, δ so that $p[C(F)] \geqslant p(C) - \mu 2^{n^{-\tau}}$.

The proof of this Lemma will be by induction, which will show that the simultaneous construction definition of the fooling partition assignment pair actually simulates the natural iterative construction. For this we need the following definition and Lemmas 4 and 5.

DEFINITION 7. For all $0 \leqslant i \leqslant \mu$ let Y_i be the random assignment to the variables in A that coincide with F on $\bigcup_{j<i} P_i$ and take random values uniformly and independently of $\langle F, P \rangle$ on $A - \bigcup_{j<i} P_i$.

LEMMA 4. For all but a fraction n^{-u-2} of the values B that P may take, and for all $0 \leqslant i \leqslant \mu - 1$ we have $|p[C(Y_i)] - p[C(Y_{i+1})]| \leqslant n^{-u-2}$, when these probabilities are conditioned by the event $P = B$.

Note that, conditioned on the event $P = B$ we have $Pr[C(Y_0) = 1] = p(C)$ and provided that $P_\mu = A_\mu$ we have $Pr[C(Y_\mu) = 1] = Pr[C(F) = 1]$ so, according to property (6) of a fooling pair, Lemma 4 implies part (1) of Proposition 3.

Proof of Lemma 4. In the following proof all probabilities are considered with the condition $P = B$. Let $F_i = F|_{\bigcup_{j<i} P_j}$. Then for every value B that P may take $Pr[C(Y_i) = 1] = \sum_f Pr(F_i = f) Pr[C(Y_i) = 1 | F_i = f]$ where f takes all of the possible values for F_i.

Suppose now that f is fixed. We may consider C as a circuit with the variables $A - \bigcup_{j<i} P_i$ by evaluating the remaining variables according to f. We will denote this circuit by D. According to the definition of a fooling pair, P_i, has the d-iterated small intersection property with parameters $1 - 2\delta$, δ, t (even if F_i is fixed). Proposition 2 implies that for all but a fraction n^{-u-4} of values B_i that P_i

may take we have that the set B_i is (t, τ)-local with respect to the circuit D. Therefore (2) of Corollary (5) implies that for all but a fraction n^{-u-3} of values B_i that P_i may take, given the event $F_i = f$ we have

$$|Pr[C(Y_i) = 1] - Pr[C(Y_{i+1}) = 1]| \leqslant n^{-u-3}.$$

Since $\mu < n$ this implies that for all but a fraction n^{-u-2} of the values B that P may take the inequality holds for all $i = 0, \ldots, \mu - 1$.

LEMMA 5. For all but a fraction n^{-u-2} of the values $B = \langle B_0, \ldots, B_\mu \rangle$ that P can take and for all $0 \leqslant i < \mu - 1$ if f is an assignment with $Pr(F_i = f | P = B) > 0$, then there exists an extension f' of f to $\bigcup_{j \leqslant i} B_j$ so that

$$Pr(F_i = f' | P = B) > 0, \text{ and}$$

$$Pr[C(Y_{i+1}) = 1 | P = B, F_{i+1} = f']$$
$$\geqslant Pr[C(Y_i) = 1 | P = B, F_i = f] - 2^{-n^\varepsilon}.$$

The proof of this Lemma is essentially the same as the proof of Lemma 4, only in the last step we use property (3) from the modified form of Corollary 5 as described in Proposition 2. Part (2) of Proposition 3 follows from Lemma 5.

Proposition 4 deals with the explicit construction of a fooling pair from a small number of random bits.

PROPOSITION 4. Let d, t be integers $\delta > 0$. Then for every large enough n a fooling pair $\langle F_n, P_n \rangle$ can be constructed with $\mu = [n^{(d+1)\delta}]$, by a LOGSPACE uniform PC family of circuits, given as inputs $2d(t+1)(\mu+1)\log_2 n + n^\delta$ random bits.

DEFINITION 8. (1) We will denote the finite field with q elements by K_q. We suppose (without the loss of generality) that n is a power of two, $n = 2^h$ and $K_n = K_2[\gamma_n]$ (where γ_n is given uniformly) and so the elements $1, \gamma_n, \ldots, (\gamma_n)^{h-1}$ form a basis of the vectorspace K_n over K_2. For each $x \in K_n$ let $\bar{x}$ denote the sequence of coefficients of the representation of x in this basis. (There is no difficulty with giving the bases of K_n in LOGSPACE since an irreducible polynomial over K_2 of degree h can be found by the brute force method in LOGSPACE.)

(2) Let $g_n(x_1, \ldots, x_{u(n)}, y_1, \ldots, y_{v(n)})$ be a function with $u(n) + v(n)$ variables where each x_i can be an element of K_n and each y_j can be an integer between 0 and 2^{h-1}. We say that g_n can be uniformly computed by a family of (n^{c_1}, c_2) circuits if there are absolute constants c_1, c_2 and a uniform family of (n^{c_1}, c_2) circuits C_n with $[u(n) + v(n)]h$ inputs and h outputs so that for any sequence $x_1, \ldots, x_{u(n)}, y_1, \ldots, y_{v(n)}$ from the domain of g if the input of C_n is $\bar{x}_1, \ldots, \bar{x}_{u(n)}, \bar{y}_1, \ldots, \bar{y}_{v(n)}$ (where $\bar{y}_j$ is the binary representation of the number y_j) the output is $\bar{g}(x_1, \ldots, x_{u(n)}, y_1, \ldots, y_{v(n)})$.

LEMMA 6. The following functions can be computed by an (n^{c_1}, c_2) circuit:

(1) $g(x_1, x_2)$ or $g(x, y)$ for any $g \in LOGSPACE$.
(2) $x_1 x_2$.
(3) x^y.
(4) $x_0 + x_1 + \cdots + x_{h-1}$.
(5) $x_0 + x_1 x_h + x_2 x_h^2 + \cdots + x_{h-1}(x_h)^{h-1}$.

Proof. (1) If $\bar{x}_1 = \langle \alpha_0, \ldots, \alpha_{h-1} \rangle$, $\bar{x}_2 = \langle \beta_0, \ldots, \beta_{h-1} \rangle$, $\bar{g}(x_1, x_2) = \langle \gamma_0, \ldots, \gamma_{h-1} \rangle$ then $\gamma_i = \vee' \wedge_{j=0}^{h-1} (\alpha_j \leftrightarrow a_j \wedge \beta_j \leftrightarrow b_j \wedge \gamma_j \leftrightarrow d_j)$ where $\vee'$ is taken for all $a, b, d \in K_n$ with $d = g(a, b)$ and $\bar{a} = \langle a_0, \ldots, a_{h-1} \rangle$, $\bar{b} = \langle b_0, \ldots, b_{h-1} \rangle$, $\bar{d} = \langle d_0, \ldots, d_{h-1} \rangle$.

(2) and (3) follows immediately from (1).

(4) If $\alpha_{i,0}, \ldots, \alpha_{i,h-1}$ are the coefficients of x_i and $\beta_0, \ldots, \beta_{h-1}$ are the coefficients of $g(x_0, \ldots, x_{h-1})$ is the basis $1, \gamma, \ldots, \gamma^{h-1}$ then $\beta_j = \vee'[(\alpha_{0,j} \leftrightarrow a_0) \wedge \cdots \wedge (\alpha_{h-1,j} \leftrightarrow a_{h-1})]$ where the disjunction $\vee'$ is taken for all 0, 1 sequences $a_0, \ldots, a_{h-1}$ with $a_0 + \cdots + a_{h-1} \equiv 1 \pmod 2$. Since there are only $n/2$ such sequences the size of the circuit is polynomial in n.

(5) follows from (4), (3), and (2).

DEFINITION 9. If $x \in K_n$ let $int(x)$ denote the integer whose binary representation is the same as the sequence of coefficients of x in the basis $1, \gamma, \ldots, \gamma^{h-1}$ [that is $\bar{x} = \overline{int(x)}$]. Conversely if y is an integer between 0 and $n - 1$ then let $fld(y)$ be the element of K_n with $int[fld(y)] = y$.

(2) Suppose i, t are integers $0 \leqslant i, t < n$. Let us define a random variable $Z_{i,t}^n = \langle z_0, \ldots, z_{n-1} \rangle$ in the following way. Take a random polynomial of degree at most $t - 1$ $f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ over K_n, so that $a_0, \ldots, a_{t-1}$ are chosen uniformly

and independently from K_n. For each $0 \leqslant j < n$ let $z_j = 1$ iff $int[fld(j)] < i$.

(3) We define a random variable $S^n_{i,t,k}$ where i, t are as in the previous definition and k is a positive integer, whose values are of $\{0, \ldots, n-1\}$. Let $\langle s^1_0, \ldots, s^1_{n-1}\rangle, \ldots, \langle s^k_0, \ldots, s^k_{n-1}\rangle$ be k independent random values of the random variable $Z^n_{i,t}$. For each $0 \leqslant j < n$ let $j \in S^n_{i,t,k}$ iff $s^1_j = s^2_j = \cdots = s^k_j = 1$.

LEMMA 7. (1) If $0 \leqslant i, t < n$ then $Z^n_{i,t}$ is an $(n, t, i/n)$-uniform random variable.

(2) If $i = n^\alpha$ then $S^n_{i,t,k}$ satisfies the k-iterated small intersection property with parameters $(1-\alpha, \alpha/2, t)$.

(3) $S^n_{i,t,k}$ has the following property: for each $0 \leqslant j < n$ we have $Pr(j \in S^n_{i,t,k}) = (i/n)^k$.

(4) If $i(n), t(n), k(n) \in LOGSPACE$ and $i(n), k(n) < n$, $0 \leqslant i < n$, $0 \leqslant t(n) \leqslant \log_2 n = h$ then there exists uniform families C_n, D_n of (n^{c_1}, c_2) circuits where c_1, c_2 are absolute constants, which realizes $Z^n_{i(n),t(n)}$ and $S^n_{i(n),t(n),k(n)}$. More precisely C_n has th inputs, D_n has kth inputs and both have n outputs, and if we take their input randomly (uniformly) then the output sequence of C_n has the same distribution as $Z^n_{i(n),t(n)}$ and the output sequence of D_n has the same distribution as the characteristic function of $S^n_{i(n),t(n),k(n)}$.

Proof of Lemma 7.

(1) See e.g. [KUW].

(2) Part (1) and Lemma 3 implies the assertion for $k = 1$. For an arbitrary k our statement follows from the trivial fact that the intersection of k independent random variables with the small intersection property with parameters $(1-\alpha, a/2, t)$ has the k-iterated small intersection property with the same parameters.

(3) For $k = 1$ our statement is equivalent to the following: if f is a random polynomial of degree at most t in $K_n[x]$ then for any fixed $b \in K_n$ $Pr\{int[f(b)] < i\} = i/n$. This assertion follows from the fact that $f(b)$ has a uniform distribution in K. For an arbitrary k our assertion follows from the case $k = 1$ and the independence of the sequences $\langle s^j_0, \ldots, s^j_{n-1}\rangle$ in the definition of $S^n_{i,t,k}$.

(4) Follows from (5) of Lemma 6, since it guarantees that the random polynomial in the definition of $Z^n_{i,t}$ can be evaluated by a (n^{c_1}, c_2) circuit.

DEFINITION 10. If C is a circuit with n outputs then let $seq(C) = \langle s_0, \ldots, s_{n-1} \rangle$ be a random variable whose values are the output sequences of C if the input of C is taken randomly and uniformly from the set of all possible inputs and let $set(C) = \{0 \leqslant j < n \mid s_j = 1\}$.

Proof of Proposition 4. Assume $A = \{0, 1, \ldots, n-1\}$. Let $\delta > 0$, d, t be fixed. Suppose that n is sufficiently large and $\mu = [n^{(d+1)\delta}]$. Let $D_0, D_1, \ldots, D_\mu$ be disjoint (n^{c_1}, c_2) circuits with $d(t+1)\log_2 n$ inputs and n outputs, so that for all $i = 0, \ldots, \mu$ the random variable $set(D_i)$ satisfies the d-iterated $(1-\delta, \delta/2, t)$ small intersection property and for any $r \in A$ $Pr[r \in set(D_i)] \geqslant (n^{1-\delta}/n)^\delta$ (where c_1 and c_2 are absolute constants). Note that Lemma 7 implies the existence of such circuits. Let $G_0, G_1, \ldots, G_{\mu-1}$ be disjoint (n^{c_3}, c_4) circuits with $(t+1)\log_2 n$ inputs and n outputs (where c_3, c_4 are absolute constants) so that for each $0 \leqslant i < \mu$ $seq(G_i)$ is (n, t)-uniform. Lemma 7 guarantees the existence of such circuits. Let G_μ be a circuit with $[n^\delta]$ inputs and n outputs so that the value of the ith output is equal to the value of the ith input if $i \geqslant n^\delta$ and 0 otherwise. Of course if the input is randomized uniformly, then we get a uniform distribution on the first $[n^\delta]$ outputs.

Now we define the circuit C that will give as an output the function F of a fooling pair.

Suppose that the circuits $D_0, \ldots, D_\mu$, $G_0, \ldots, G_\mu$ are pairwise disjoint. If $p_0, \ldots, p_\mu, q_0, \ldots, q_\mu$ are inputs for $D_0, \ldots, D_\mu, G_0, \ldots, G_\mu$ then $p = \langle p_0, \ldots, p_\mu, q_0, \ldots, q_\mu \rangle$ will be an input for C. C will have n outputs and the value of the ith output $C(p \mid i)$ is defined by the following boolean expressions. If $i \geqslant [n^\delta]$ then

$$C(p \mid i) = \bigvee_{j=0}^{\mu-1} [G_j(q_j \mid i) \wedge D_j(p_j \mid i) \wedge \bigwedge_{r=0}^{j-1} \neg D_r(p_r \mid i)] \qquad (*)$$

otherwise $C(p \mid i) = G_\mu(q_\mu \mid i)$.

[The meaning of $(*)$ is the following: if j is the smallest non-negative integer with $j < \mu$ and $D_j(p_j \mid i) = 1$ then $C(p \mid i) = G_j(q \mid i)$, if there is no such integer then $C(p \mid i) = 0$].

Clearly there is an (n^{c_5}, c_6) circuit with these properties where c_5, c_6 are absolute constants.

We define the parts of $P_0, \ldots, P_\mu$ in the partition as follows: thinking of $D_0, D_1, \ldots, D_{\mu-1}$ as subsets of A, we take P_j, $j < \mu$, to be all elements in D_j that do not belong to D_r, $r < j$, and P_μ are the

remaining elements. Formally, for $j < \mu$

$$i \in P_j \leftrightarrow i \geqslant n^{\delta} \wedge D_j(p_j \mid i) = 1 \wedge \bigwedge_{r=0}^{j-1} D_r(p_r \mid i) = 1 \quad \text{if}$$

$$P_\mu = A - \bigcup_{j<\mu} P_j.$$

Now we prove that $\langle F, P \rangle$ is a fooling pair.

(1) follows immediately from the definition of F.

(2) The defining formula of P_j implies that the sets are disjoint and the definition of P_μ implies that they cover A.

(3) For $i < \mu$ $P_i = set(D_i) - \wedge_{j<i} set(D_j) - \{r \mid r < n^{\delta}\}$. The conditions $P_0 = A_0, \ldots, P_{i-1} = A_{i-1}$ restrict only the values of $D_0, \ldots, D_{i-1}$ but D_i is independent of them. Therefore $set(D_i)$ has the d-iterated small intersection property with parameters $(1 - \delta, \delta/2, t)$ even with the conditions $P_0 = A_0, \ldots$. Since $P_i \subseteq set(D_i)$ it also has the same small intersection property.

(4) If $P_j = A_j$ then $F|_{A_j} = seq(G_j)|_{A_j}$. Since the outputs of each G_j are independent for $j = 0, \ldots, i$ the random variables $F|_{A_0}, \ldots, F|_{A_i}$ are also independent.

(5) If $P_i = A_i$ then $F|_{A_i} = seq(G_i)|_{A_i}$ and $seq(G_i)$ is (n, t)-uniform and as we have shown in the proof of (3) it is independent of P_i.

(6) Let $A_\mu = \{r \mid r < n^{\delta}\}$. $A_\mu \subseteq P_\mu$ always holds according to the definition of A_μ and P_μ. The definition of G_μ and $F|_{A_\mu} = seq(G_\mu)|_{A_\mu}$ implies that $F|_{A_\mu}$ has a uniform distribution so we have to prove only $Pr(P_\mu = A_\mu) \geqslant 1 - 2^{-n^{\delta}}$.

Let $r \in A - A_\mu$, $i < \mu$ be fixed. According to our assumption $Pr[r \in set(D_i)] \geqslant n^{-d\delta}$. Since the D_i's are independent we have $Pr[r \notin \bigcup_{i<\mu} set(D_i)] \leqslant (1 - n^{-d\delta})^{\mu} \leqslant (1 - n^{-\delta d})^{n^{(d+1)\delta}} \leqslant 2^{-n^{\delta}}$. This proves our assertion since $P_\mu \cap \bigcup_{i<\mu} set(D_i) = 0$.

The Main Theorem follows from Proposition 3 and Proposition 4 if we choose any $\delta > 0$ so that $\delta \to 0$ as $n \to \infty$.

5. DEPTH 2 CIRCUITS

The two results in this section are algorithms for the problem of approximate counting and finding a satisfying assignment,

respectively, in depth 2 circuits (or CNF/DNF formulas). The two important parameters that affect the running time of the algorithms are the fraction of satisfying assignments and the sizes of clauses.

Let $A = \{x_1, \ldots, x_n\}$ be a set of boolean variables. A clause C is a conjunction of literals from A, e.g., $C = x_1 \wedge \bar{x}_3 \wedge x_6$. A DNF formula F is a disjunction of clauses (we will take F to be both the set of clauses and their disjunction, so $F = \vee_{C \in F} C$). $|F|$ denotes the number of clauses in F. For a clause or set of clauses H, $v(H)$ will denote the set of variables occurring in H. If for all $C \in F$, $|v(C)| \leqslant k$, then F is a $k - DNF$ formula. Similarly we define k-CNF formula.

We need some notation which is similar to that of Section 4.

Assume $A \subseteq v(F)$, and $Y \subseteq A$. We can restrict F by assigning values to variables in Y. If $y \in \{0,1\}^{|Y|}$, then $F_{Y,y}$ denotes the restricted formula after assigning y to Y (in order). Say that restrictions (Q, q) and (Y, y) satisfy $(Q, q) \geqslant (Y, y)$ if $Y \subseteq Q$ and q agrees with y on Y.

We further define F_Y to be the random variable (formula) $F_{Y,y}$ where $y \in \{0,1\}^{|Y|}$ is chosen uniformly at random. Note that if $Z \subseteq Y$ then $Pr(F_Y \not\equiv 0, 1) \leqslant Pr(F_Z \not\equiv 0, 1)$.

Theorem 1 deals with the approximation of depth 2 circuits. It shows that the output almost always depends on a small subset of the input variables.

THEOREM 1. Let C be a depth 2 circuit of bottom fan-in $\leqslant k$ with input variables in A. Then for every $r \geqslant 2^{2k^2}$ there exists a subset $Q \subseteq A$ s.t.

(1) $|Q| \leqslant r^{2k^2}$.
(2) $Pr(C_Q \not\equiv 0, 1) \leqslant 2^{-r}$.

Furthermore, if $|A| = n$, the set Q can be found in time $O(n^{k^2})$.

Proof. It is enough to prove Theorem 1 in the case when the boolean formula F corresponding to C is a k-DNF formula. We prove the theorem by induction on k.

$k = 1$. If $|F| \leqslant r$ then set $G = F$ else let G be a subset of any r clauses in F. Let $Q = v(G)$. Then $|Q| \leqslant r \leqslant r^2$ and $Pr(F_Q \not\equiv 0, 1) \leqslant 2^{-r}$.

$k > 1$. Assume the inductive assumption for all values less than k. Let G be a maximal subset of pairwise disjoint clauses from F formally $E, D \in G \rightarrow v(E) \cap v(D) = \varnothing$ but for all $E \in F - G$ there exist a $D \in G$ with $v(E) \cap v(D) = \varnothing$.

Case 1: $|G| > r2^k$. Let $\bar{G} \subseteq G$ with $|\bar{G}| = r2^k$. Set $Q = v(\bar{G})$. We have $|Q| \leqslant kr2^k \leqslant r^{2k^2}$, and

$$Pr(F_Q \not\equiv 0,1) \leqslant Pr(F_Q \not\equiv 1) \leqslant \prod_{E \in \bar{G}} Pr(E_Q \not\equiv 1) \leqslant (1 - 2^{-k})^{|\bar{G}|}$$

$$\leqslant (1 - 2^{-k})^{r2^k} \leqslant 2^{-r}.$$

Case 2: $|G| \leqslant r2^k$. Let $Z = v(G)$. Partition the clauses in $F - G$ into families, $H(Y, y)$ one for each $Y \subseteq Z$, $1 \leqslant |Y| \leqslant k - 1$, and $y \in \{0,1\}^{|Y|}$, as follows: $H(Y, y) = \{E \in F - G | v(E) \cap Z = Y$ and $E_{Y,y} \not\equiv 0\}$. Clearly there are at most $(kr2^k)^k$ such families, and $F \equiv G \vee \bigvee_{Y,y} H(Y, y)$.

Consider the formulas $H_{Y,y} = H(Y, y)_{Y,y}$. $v(H_{Y,y}) \subseteq A - Z$, and each is a $(k-1) - DNF$ formula since G was maximal. Apply the inductive assumption to each $H_{Y,y}$ with parameters $k - 1$ for k and $2r$ for r. Let $Q_{Y,y}$ be the sets guaranteed inductively. Hence for all Y, y we have

(1) $|Q_{Y,y}| \leqslant (2r)^{2(k-1)^2}$ and
(2) $Pr[(H_{Y,y})_{Q_{Y,y}} \not\equiv 0,1] \leqslant 2^{-2r}$.

Now set $Q = Z \cup \bigcup_{Y,y} Q_{Y,y}$. Then
(1) $|Q| \leqslant |Z| + \sum_{Y,y} |Q_{Y,y}| \leqslant rk2^k + (rk2^k)^k (2r)^{2(k-1)^2} \leqslant r^{2k^2}$.
To prove that $Pr(F_Q \not\equiv 0,1) \leqslant 2^{-r}$ we observe the following. Let $q \in \{0,1\}^Q$ s.t. $F_{Q,q} \not\equiv 0,1$. Then $G_{Q,q} \equiv 0$, and in fact, $F_{Q,q} \equiv \bigvee_{(Q,q)>(Y,y)} (H_{Y,y})_{Q,q}$. Therefore for at least one pair (Y, y), $(H_{Y,y})_{Q,q} \not\equiv 0,1$, and since $Q_{Y,y} \subseteq Q$ we have
(2) $Pr(F_Q \not\equiv 0,1) \leqslant \sum_{Y,y} Pr[(H_{Y,y})_{Q,q} \not\equiv 0,1] \leqslant (rk2^k)^k 2^{-2r} \leqslant 2^{-r}$.

The proof shows that the subset Q can be found in $DTIME(n^{k^2})$. [Indeed, let $h(k)$ denote the time necessary for finding Q. Since $|F| \leqslant n^k$, G can be found in time $O(n^k)$. $Q = Z \cup \bigcup_{Y,y} Q_{Y,y}$, which implies that Q can be found in time $O(n^k) + \sum_{Y,y} h(k-1) \leqslant n^k h(k-1)$ that is $h(k) \leqslant n^k h(k-1)$.]

Theorem 1 can be used as an algorithm for approximate counting (see Section 3.1).

COROLLARY 6. Let k, p, β be fixed, F any k-CNF or k-DNF formula, $p(F) = p$. Then the β-approximation problem can be solved in deterministic polynomial time.

Theorem 2 gives a somewhat faster algorithm for the simpler problem of finding a satisfying assignment.

THEOREM 2. Let F be a satisfiable k-CNF formula on n variables with $p = p(F)$. Then we can find a satisfying assignment of F in $DTIME(k|F| + 2^{k2^k(\log p^{-1})})$.

For example this theorem says that 3-CNF instances of SAT with a polynomially small fraction of satisfying assignments are easy, as we can find one in polynomial time!

Proof. (Sketch). Simple counting shows that any maximal set of clauses has at most $2^k(\log p^{-1})$ elements, otherwise some clauses would be satisfied. We try all assignments to this variable and proceed with induction on k.

REFERENCES

[Ad] L. Adleman, "Two theorems on random polynomial time," *19th FOCS* 75–83 (1978).

[Aj] M. Ajtai, "Σ_1^1-formula on finite structures," *Ann. Pure Appl. Logic* 24: 1–48 (1983).

[An] R. Anderson, "Set splitting," manuscrupt.

[AB] M. Ajtai and B. Ben-Or, "A theorem on probabilistic constant depth computations," *16th STOC* 471–474 (1984).

[ACGS] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, "RSA/Rabin bits are $1/2 + 1/poly(\log n)$ secure," *25th FOCS* 449–457 (1984).

[AW] M. Ajtai and A. Wigderson, "Deterministic simulation of probabilistic constant depth circuits," *26th FOCS* 11–19 (1985).

[BBS] L. Blum, M. Blum, and M. Shub, "A simple secure pseudo-random number generator," *Adv. Cryptogr. Proc. CRYPTO-82* 61–78 (1982).

[BM] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo random bits," *23rd FOCS* 112–117 (1982).

[ES] P. Erdös and J. Spencer, *Probabilistic Methods in Combinatorics*. Academic Press, New York, 1974.

[FLS] M. Furst, R. J. Lipton, and L. Stockmeyer, "Pseudo-random number generation and space complexity," *Information Control*, to appear.

[FSS] M. Furst, J. B. Saxe, and M. Sipser, "Parity circuits and the polynomial time hierarchy," *22nd* 260–270 (1981).

[Gi] J. Gill, "Complexity of probabilistic Turing machines," *SIAM J. Computing* 6: 675–695 (1977).

[GG] O. Gaber and Z. Galil, "Explicit construction of linear sized superconcentrators," *J Comp. Sys. Sci.* 22: 407–420 (1981).

[Ha] J. Hastad, "Lower bounds for the size of parity circuits," manuscript.

[KUW] R. M. Karp, E. Upfal, and A. Wigderson, "A fast parallel algorithm for the maximal independent set problem," *24th STOC* 266–272 (1984).

[KW] R. M. Karp and A. Wigderson, "A fast parallel algorithm for the maximal independent set problem," *24th STOC* 266–272 (1984).

[Lu] M. Luby, "A simple parallel algorithm for the maximal independent set problem," *17th STOC* (1985).

[Ma] G. A. Margulis, "Explicit construction of graphs without short cycles and low density codes," *Combinatorica* 2(1): 71–78 (1982).

[Ru] W. L. Ruzzo, "On uniform circuit complexity," *J. Comput. Sys. Sci.* 22(3): 365–383 (1981).

[RT] J. H. Reif and J. D. Tygar, "Towards a theory of parallel randomized computation," TR-07-84, Aiken Computation Lab., Harvard University, 1984.

[Si] M. Sipser, "A complexity theoretic approach to randomness," *15th STOC* 330–335 (1983).

[Sh] Shamir, "On the generation of crytographically strong pseudo random sequences," *8th ICALP*, Lecture notes in *Comp. Sci.* 62: 544–550. Springer-Verlag, Berlin, 1981.

[St] L. Stockmeyer, "The complexity of approximate counting," *15th STOC* 118–126 (1983).

[VV] L. G. Valiant and V. V. Vazirani, "NP is as easy as detecting unique solutions," *17th STOC* (1985).

[Ya] A. C. Yao, "Theory and applications of trapdoor functions," *23rd FOCS* 80–91 (1982).

SELF-CORRECTING TWO-DIMENSIONAL ARRAYS

Peter Gacs

ABSTRACT

We continue the program begun in two earlier works: to find simple and efficient ways to implement computations of arbitrary size by a homogeneous array of unreliable elementary components. The homogeneity of cellular arrays makes the *maintenance* of the error-correcting *organization* the biggest technical problem. This problem could be completely avoided in the earlier three-dimensional construction, where almost no structure was needed. All error correction was performed using Toom's voting rule. The structure maintenance problem dominates the earlier one-dimensional construction since there is no Toom's Rule in one dimension. Here, we present a two-dimensional solution, desirable for several reasons. Information maintenance is no longer possible by Toom's Rule, forcing us to adopt a hierarchical design. But Toom's Rule is still applicable to

Advances in Computing Research, Volume 5, pages 223–326.

ISBN: 0–89232–896–7

structure maintenance, giving a significant complexity-dimension trade-off. The second achievement of the present work is the constant space redundancy. Despite the noise in its elementary components, our computing device has a positive capacity as a "communication channel" from input to output. Great care was given to terminology and structured presentation to create a reference point on which further developments (e.g., the introduction of asynchrony) can be built. The first third of the paper serves as an informal overview of the subject.

1. ASSUMPTIONS

There are some general assumptions underlying our investigation that distinguish it from many other studies addressing problems of fault-tolerant computing. These assumptions are as follows.

- We have to build computing devices of *arbitrary size* from a few types of *elementary components*.
- Faults occur in each component *independently* at different space–time positions. Faults thus do not conspire: only the computation creates dependence between the results of faults.
- Faults are transient: they change the local state but not the local transition rule.
- The probability of component faults is bounded by some small parameter ϱ independent of the size of the computing device.
- There are no other restrictions on the type of fault that can occur. Thus, faults can cause arbitrary local state changes.

The goal is to find architectures coping with all combinations of faults likely to arise for devices of a given size. We also want to estimate the needed sacrifice in time and memory to make a computation reliable.

1.1. Transient Faults

The requirement that the faults be transient restricts us severely. This is the kind of fault that is only a one-time violation of the rule of operation. It does not incapacitate the component permanently in which it occurs. In the following steps, the component obeys its rule of operating again, until the next fault.

The toughness of the condition becomes clear if we try to *implement* each component of a theoretical device by a small network of transistors. In a typical implementation, the states of the component are mapped into *some* configurations of the network. But most configurations will be illegal. Therefore the fault of one transistor brings the network probably into an illegal configuration not representing any state of the implemented component. It is better therefore to understand under implementation something like a *computing crystal*: a compound whose transition rules are determined by physical law. It is unlikely, of course, that some chemist will find or synthetize a compound with exactly the same transition laws that we propose. Our constructions must be viewed as *existence proofs*. However, they help to clarify what properties of the local transition rule are needed for reliable computation.

Transient faults are not an exotic kind. Their main source is thermal noise that becomes more significant as the scale of physical devices goes down.

The case of nontransient component fault is of practical importance, but it has not been investigated in the same generality. There are reasons to believe that many of the techniques developed for the case of transient faults will be applicable to the case of permanent faults.

1.2. Parallel Architectures

A reliable computer, all of whose components are unreliable, must use massive parallelism. Indeed, information stored anywhere during computation is subject to decay and therefore must be actively maintained.

Problems concerning serial computers are of less "ideological purity," since they must except a part of the device from the influence of faults. Still, the following problem seems technically very challenging. Manuel Blum asked whether reliable computation is possible with Turing machines in which only the internal state is subject to faults: the tape is safe. Presently, it seems that the solution of this problem requires a construction analogous to the one given in [G].

1.2.1. Inhomogeneous Devices

von Neumann [VN] designed a Boolean (acyclic) circuit that works reliably even if its components are unreliable. In his model,

each component had some constant probability of fault. For a deterministic circuit consisting of N components, he built a circuit out of $O(N \log N)$ stochastic components, computing the same function with large probability. (For an efficient realization of his ideas, see [DO2].) The key element of his construction is a *restoring unit*. This unit is supposed to *suppress the minority*. If among its n Boolean input values fewer than $0.3n$ are different from the majority then among the n Boolean output values, fewer than $0.1n$ values differ, even if each component of the restoring unit fails with probability 0.01. The restoring unit of [DO2] achieving this consists of several layers of majority gates. The inputs for each such gate are three randomly selected sites of the previous layer.

The redundancy factor $\log N$ in von Neumann's construction seems to be essential. The number N can be called the *size* of the computation. In other models of computation, it corresponds to the number of elementary events (including information storage) during the whole computation, i.e., roughly to the product of the number of components by the number of computational steps. This estimate is a little weak if the only goal of the computation is information storage.

For purposes of information storage, it is better to use the model of *clocked circuits*. These are Boolean circuits in which special one-bit memory elements called *shift registers* are also permitted. Cycles are permitted but each cycle must contain at least one shift register.

We cannot store even one bit in n registers of such a circuit for longer than 2^n steps, since during that time quite probably a step will occur in which the content of all cells changes simultaneously. But the paper [Kuz] showed that once we have n registers, it is possible to store there $c \cdot n$ bits for 2^{cn} steps, where the positive c depends only on the fault probability of the individual components. This work used Gallagher's low-density parity-check codes, as proposed in [Ta], but improved both in performance and proof by Pinsker.

1.2.2. *Cellular Automata*

The above constructions suffer from the same *theoretical flaw*: the circuits use a rather intricate connection pattern that cannot be realized in three-dimensional space with wires of constant length. On the other hand, the natural assumption about a wire is that as its length grows, its probability of fault converges to 0.5.

A cellular space (medium) is a lattice of automata in, say, three-dimensional space where every automaton takes its input from a few of its closest neighbors. First introduced by von Neumann and Ulam, such devices are now sometimes known as "systolic arrays," or "iterative arrays." Mathematical physicists use the more general term *interactive particle system.*

Typically, all automata are required to have the same transition function and are connected to the same relative neighbors, i.e., the device is translation invariant. The spatial uniformity suggests the possibility of especially simple physical realization. It reduces the number of ingredients that are assumed unchangeable by faults to just one: the transition rule.

Cellular media are desirable computing devices, and it is easy to construct a one-dimensional cellular space that is a universal computer. Let us indicate how a one-tape Turing machine can be simulated by a cellular array, since these machines are better known to be universal.

A Turing machine is defined to have an infinite array of tape cells and a head. Each cell can be in a finite number of possible states, and so can the head. In each instant, the head is scanning a cell. Depending on the state of the scanned cell and its own state, the head makes one of three possible movements (left, right, stay), and changes its own state and the state of the scanned cell. The rule of operation of the machine describes this dependence.

In a possible simulation, the cellular array simulating the Turing machine has one cell corresponding to each tape cell of the Turing machine. A state of a new cell indicates the state of the represented tape cell, the fact whether the head is scanning the tape cell and if yes, what is the head's state. The transition rule of this simulating array is easy to derive from the transition rule of the simulated Turing machine.

2. PHILOSOPHICAL DIGRESSION

2.1. Notions of Complexity

Our remarks make use of several very different notions of *complexity*, and it will be important to keep them apart. Some of these notions have now very adequate mathematical definitions, while some other ones do not.

2.1.1. *Descriptional Complexity*

Kolmogorov and Solomonoff introduced the notion of *descriptional complexity*. In its current version (see [Le]), the complexity $K(x)$ of a finite string can be defined as the length of the shortest self-delimiting program needed to direct some fixed univeral computer to output x. This complexity could be also called "information content," or "measure of disorder," "measure of randomness," or "individual entropy." According to this notion, a typical string of n bits arising from a coin-tossing experiment is maximally complex. Let $\mathrm{Prob}_t(x)$ be the probability to obtain x as output on some fixed universal Turing machine [the same one used in the definition of $K(x)$] in t or fewer steps from a coin-tossing input. The relation $K(x) = -\log \mathrm{Prob}_\infty(x) + O(1)$ is known (see [Le]).

2.1.2. *Computational Complexity*

In the Theory of Computation, another notion of complexity has widespread use. This complexity is the property of a function. The function $g(x)$ is a lower bound on the time complexity of function $f(x)$ on Turing machines if for each Turing machine T there is a positive constant c such that for all x, T takes at least $cg(x)$ steps to compute $f(x)$.

2.1.3. *Logical Depth*

Descriptional complexity helped define individual information and individual randomness. It also helps define a notion that can be considered the "individual computational complexity" of a finite object. Some objects are clearly the result of long development (computation), and are extremely unlikely to arise by any probabilistic algorithm in a small number of steps. This property of objects was formally defined by Bennett as follows (see [B]).

$$\mathrm{depth}_\varepsilon(x) = \min\{t\colon \mathrm{Prob}_t(x)/\mathrm{Prob}_\infty(x) \geqslant \varepsilon\}.$$

Thus, the depth of a string x is at least t with confidence $1 - \varepsilon$ if the conditional probability that x arises in t steps *provided it arises at all* is less than ε.

2.1.4. *Conceptual Complexity*

There is also an informal notion of *conceptual*, or *organizational complexity* that will certainly be applied to the present paper to some extent, and to [G] to a great extent. It will probably never be formally defined, though it may be related to an informal version of $\text{depth}_s(x)$ defined above, with the length of x too small to be amenable to formal treatment.

2.2. Noise Resistance and Biological Organization

Even serious scientists indulge sometimes in speculations about "what is life." The goal is to formulate some abstract criteria that among the existing structures in nature, only biological (or, possibly, social) systems satisfy. It is not easy to find such criteria: self-reproduction is clearly insufficient. Answers to this type of question may influence our perspective on some of the more immediate problems of science.

Work on fault-tolerant computation led us to the conjecture that the requirement to *perform in a noisy environment* forces rather elaborate structures, with some resemblance to the ones in biological systems. Two extremal cases of the "performance" we have in mind are *information storage* and the *production of depth*.

Much of the apparent conceptual complexity of biological systems might be due to the simple requirement of reliable information storage. Here is an informal argument in favor of this (informal) thesis: without systems of biology or human society, there seems to be no method to save 300 bits of information from decay somewhere in the universe for 10^9 years. But life has preserved much more information in the DNA for longer.

A similar thesis could be stated about systems starting with zero information but producing structures of ever increasing logical depth, in a noisy environment. Again, only systems involving biology or society seem to be capable of this.

The above general observations are in contrast with most early experimental and theoretical work on cellular automata. The physical investigations were concerned with systems whose convergence to equilibrium could be taken for granted, and only the properties of these equilibria were of interest. The computational applications built structures with logical depth. Examples are algorithms involving systolic arrays, or the structures to simulate

arbitrary computations by simple rules like Conway's Game of Life [BCG] or Margolus's Billiard Ball rule [M]. However, if faults are permitted then all structures built in these computational applications turn rapidly into shallow chaos.

We are interested to find out what properties of the elementary building blocks can help counteract chaos.

2.3. On the Need for Proofs

We deal with media whose reliability can be proven mathematically. Nobody suggested yet candidates for reliable media found by experimentation. Simple reliable—or almost reliable—media without proofs probably exist: such media might underly biological systems. But elementary constructions coupled with rigorous elementary mathematical analysis yielded more information for the present research than experimentation.

One reason (suggested by the reviewer) for the lack of more experimentation with error-correcting schemes in the model we are considering is that the fault probabilities of the elementary components are generally very small, even if constant. Experiments could therefore take a long time before an interesting effect turns up.

3. A THREE-DIMENSIONAL RELIABLE MEDIUM

3.1. Toom's Results

The first questions about the possibility of storing information in arbitrary interactive particle systems were asked in the context of *ergodicity*. Without giving a formal definition here, we recall that an ergodic system converges to a unique statistical equilibrium independent of the starting configuration. Such a system therefore cannot store even *one bit* of information, and cannot increase depth too much.

It was a nontrivial result of Toom that there are infinite two-dimensional cellular media that can store reliably a bit of information. Let us assume that in the initial configuration, each cell has the same state s. We try to limit the probability of a state different from s in each cell at all later steps. It is not easy to find "voting" rules that achieve this, and even if we find one, it is not easy to prove that it works. This is what Toom did; moreover, he characterized all such *monotonic* rules.

For the rest of the paper, we choose one rule in the above family. To determine the state of a cell in the next step, this rule takes the majority of its northern, southwestern and southeastern neighbors. In what follows we will refer to this rule simply as *Toom's Rule.*

Toom's proof in [Too] uses an elaborate topological argument that we could not adapt to an efficient finite version of the theorem. A new proof technique, the technique of "k-noise," was developed in [G] and [GR]. It gave a more straightforward proof of Toom's theorem with efficient finite implications. It is also the only probabilistic tool of the present paper. Mathematical physicists point out that it belongs to the broad class of techniques known under the name *renormalization.*

3.2. Application to Computation

Let D be any one-dimensional medium (cycle) of size K working for L steps. We can simulate its work reliably using a three-dimensional medium D' on a torus of size $K \times d \times d$ where $d = \log^{1+\varepsilon}(KL)$. Each trajectory $x[t,n]$ of D (the function giving the state of cell n at time t when the medium evolves according to the rule D) is mapped into a trajectory $z[t,n,i,j]$ of D' by

$$z[t,n,i,j] = x[t,n]. \tag{3.1}$$

Thus, each symbol of D is repeated over a whole two-dimensional slice of D'. In each step, the transition rule of D' uses first Toom's Rule within the slices then rule D across the slices. Without errors, (3.1) holds for all t,n,i,j. It is shown in [GR] that with errors, the same relation will still hold with large probability. The space requirement of the reliable computation, compared to the original one, is increased by the factor d^2. The medium D' will simulate D reliably step-for-step, without any time delay.

This simulation is the simplest design for reliable computation ever proposed. Its simplicity seems to depend on some special circumstances, especially on a too high number of dimensions, and on synchrony. When these do not hold, we have to rely on much more complicated models.

No attempt was made in [GR] to maximize the error probability permissible for reliable computation. Bennett's experiments on Toffoli's Cellular Automata Machine give convincing empirical evidence that Toom's medium is nonergodic at error probabilities

below 0.05. Piotr Berman and Janos Simon, using Toom's original proof, brought error bound needed for the proof to 10^{-7} (see [BS]).

3.2.1. *Heat Production*

There are some physical reasons to look for lower dimensional error-correcting media. A three-dimensional error-correcting medium is thermodynamically unrealizable. Indeed, any error-correcting operation is inherently irreversible. Such an operation turns a certain minimum amount of free energy into heat. Therefore each cell needs a whole private "supply line" for feeding it with free energy and relieving it of the produced heat. This is possible only if the medium is at most two-dimensional.

3.2.2. *Synchrony*

Toom's Rule keeps a bit of information in a two-dimensional medium even if the cells do not all fire at the same time. But the three-dimensional computing medium defined above is heavily dependent on the synchronous operation of all cells within a cross section. All simple synchronization techniques that we are aware of interfere with error-correction.

4. HIERARCHY OF SIMULATIONS

4.1. Coding

Reliable computation always requires coding with redundancy. Without it, part of the input is lost in the first step, and part of the output in the last step. During the computation therefore, information must live in encoded form.

In general, reliability seems to require that we break up the task into manageable pieces in space and time, and, with each subtask, we go through a cycle of decoding, computation and encoding. Each cycle is repeated several times, since an error can ruin the work of the whole cycle.

In the three-dimensional case, we enjoyed the exceptional situation that the orthogonality of the computation to the repetition code permitted immediate parallel access to each bit of the code. The repetition needed to counter errors in the computation happens to

be identical to the repetition used as redundancy in coding. Due to the lack of decoding or transport, no special structures are required to support the error-correcting activity.

In the general case, information needs to be transported from storage to the place where it is processed. There, it must be decoded. After processing, it must be transported back for storage. All this needs elaborate organization devoted to the task of error correction. In the homogeneous media we are discussing now, this organization can exist only as "software," just as perishable as the rest of information, and it needs maintenance.

The program also must have some *continuity property*: small errors have small consequences. This property will be achieved by insisting that in any one phase of the program, only a small part of the information can be changed. To enforce this requirement, it will be necessary to keep track of phase and relative position by especially robust methods.

4.2. Increasing Reliability by Repeated Decoding

In a computation of size N there will be arbitrary groups of faults of size of the order of $\log N$. Reliable computation must be organized in such a way that no crucial piece of information is committed to a number of cells smaller than $\log N$. Whatever coding we use it must operate with units greater than $\log N$. The tasks of decoding, shipping, etc. with these units are large-scale computational tasks in themselves that must be organized reliably.

The greater units can be derived mathematically only from the primitive "physical" cells of our original medium M, using decoding. Decoding transforms the original probability distribution in M into a different one, in which the great units will operate much more reliably than the original cells.

We could view the greater units as a new kind of cell with a large state space (always as large as needed). It is more advantageous to view them as *colonies* built of the cells of a standard universal medium Univ. From colonies of a given size P, we will arrive at colonies of a larger size P^* and higher reliability in the following way. We choose a certain parameter Q, group the colonies in arrays of size Q by Q (called *alliances*), and apply the decoding of a certain error-correcting code to the configuration found in the resulting array of size QP by QP.

Hierarchy was used in the context of inhomogeneous one-dimensional cellular arrays in [Ts]. The transition rule of the cells was required to vary hierarchically in space and time, to organize a voting scheme, with the sole purpose of remembering one bit of information.

The paper [Kur] made a rather elaborate proposal for such a hierarchy of simulations. Several ideas of this proposal were used in [G].

5. MAINTENANCE OF THE ORGANIZATION

5.1. Structure Maintenance

Above, we referred to a hierarchy of simulations that we had to use for reliable computation in dimensions 2 and 1. On each level of the hierarchy, for some parameters P, Q, P^*, colonies of size P ("small colonies") are grouped in a Q by Q array (the alliance) to simulate a colony of size P^* (a "big colony"). All these configurations are in the universal medium Univ. Each small colony has some work period T, and the alliance has a work period TU.

5.1.1. Health

Each small colony will, of course, have to obey a certain program. In order for the program to have the desired effect, the configuration of the colony must be somewhat standardized. The standardization need not be very elaborate: we will just mark the boundaries of the colony at the beginning of its work period. We will say in this case that the colony is *healthy*. If health is lost then it seems that we cannot keep our view elevated from the alliance to the colony simulated by it, since the small colonies no longer work by their original program.

5.1.2. Legality

Even if the members of the alliance are healthy they must share some information that is of no use for the decoder but is needed for the orchestration of their work. An idea that reduces the structure maintenance problem to a manageable size is that all the

information needed for this purpose falls into one of the following categories.

- *Parameters* of the alliance: some constants that are the same for each small colony;
- Two remainders mod Q describing the position of each small colony in the alliance;
- A remainder mod U describing the current clock value in the work period of the alliance.

The last two pieces of information are called *phase variables*. The activity of the colony can be made very predictable, provided that the value of the phase variables is correct, even if the rest of the information it handles is in doubt. In particular, the colony can consult the phase variables every time it is about to write to certain places—to make sure it writes only when the program permits this. This reliance on the phase variables will guarantee the continuity property mentioned in Section 4.1. We will call a healthy small colony *legal* when its parameters and phase variables have the required values.

In some sense, the values of the parameters and phase variables are not information. Indeed, they are a simple periodic function of space and time. Legality can therefore be restored by methods more local than those needed for the rest of the information. This observation solves the problem completely for two dimensions. Healthy colonies will maintain or restore their legality by Toom's Rule.

As a result, the two-dimensional reliable medium built in the present paper has the following property:

> Legality of colonies of the lowest order will be restored over an arbitrarily large damaged area if only the frequency of faults remains small in each of their alliances. This will happen even if none of the higher order colonies is healthy.

There is no Toom Rule in one dimension. Structure maintenance is still possible by more complicated methods, as shown in [G]. That they are more complicated can be seen from the fact that the above property will no longer hold. Legality and health can be defined in one dimension similarly to the way they were defined above. If the legality of colonies of the lowest order is damaged in an area needed for simulating a colony of order k then, even without noise, it will

be restored only if outside this area, still healthy colonies of order k are simulated. Thus, in two dimensions, to restore the legality to lowest order colonies in a damaged area, it was enough if it was surrounded with an area of comparable *size* containing legal colonies. In one dimension, not only the size of the surrounding area matters: it must be of comparable size and *organized to the maximum level*, even if only the restoration of the legality of the smallest colonies is required.

5.2. Health Maintenance

Toom's Rule restores legality to healthy small colonies. It is not clear yet how will health be restored to damaged small colonies, in order for them to be able to apply Toom's Rule. However, if we assume that health is restored by "magic" then we can *program* the same kind of magic into the simulation of big colonies. Indeed, at the beginning of the simulation of a big colony by an alliance, the latter will cast the big colony into the standard initial form required by health.

Because of the initializing step, the simulation of colonies by alliances is not like ordinary simulation: periodically, a certain structure is *forced* on the simulated colonies, whatever their previous state was.

Essentially, this is the method also used in [G], but there, setting all parameters had to be part of the initialization. This type of organization resembles that of biological systems: each cell has the genetic code of the whole organization.

The health of colonies of the lowest order will be assured because these colonies will be single cells of the reliable medium M and, as such, they will be defined to have only healthy states.

At the end of the computation, whatever information is in the alliance must be cast in the form of a redundant code. Due to heightened requirements of efficiency, this problem is harder to solve here than it was in [G].

6. MINIMUM REDUNDANCY

All past and present results discussed in this chapter indicate that if the size of the computation is N then the size of the simulating computation must be at least $N \log N$, *provided that some real computation is being performed.* The last qualification is necessary

since information storage needs only a constant factor in redundancy. In the model of cellular arrays, it is possible to distinguish between the time and space requirements of the computation, and it is therefore possible to represent the redundancy as a product of the redundancies in space and time. Thus if t steps of computation of n cells of a deterministic medium are simulated reliably by t' steps of n' cells of a stochastic medium then t'/t is the time redundancy and n'/n is the space redundancy.

The results available to date on cellular arrays indicate that the product of the time and space redundancies must be logarithmic in the size of the computation. We can state this as a conjecture, but it seems a difficult one to prove, especially that the case of "no real computation" must be excepted.

The logarithmic redundancy can be shifted entirely to space, or almost entirely to time, as the following examples show.

- The three-dimensional simulation in [GR] is real-time: there is no time redundancy, but there is space redundancy of size $\log^{2+\varepsilon} N$ (reduced to $\log^2 N$ in [BS]). The ideas of [GR] applied to [G] give a real-time two-dimensional reliable simulation with space redundancy $\log^2 N$.
- The two-dimensional simulation described in the present chapter has constant space redundancy, and time redundancy $\log^{2+\varepsilon} N$.

7. THE CONCEPTS AND THE RESULT

7.1. Sites and Events

Let $\mathbf{Z}_m$ be the group of remainders mod m if m is finite, and the set $\mathbf{Z}$ of integers if m is infinite. The set $\mathbf{W}$ of *sites* of our cells will be $\mathbf{Z}_m^2$. If $m = \infty$ then $\mathbf{W}$ is the two-dimensional rectangular lattice $\mathbf{Z}^2$. Otherwise, it is a lattice over a torus of diameter m. As a notational convenience, we define, in analogy with Pascal and real analysis, the intervals

$$[a\,..\,b) = \{x \in \mathbf{Z}\colon a \leqslant x < b\}.$$

The following definitions are given for two dimensions but they can be generalized to any number of dimensions. For a set G and a

vector v in $\mathbf{Z}_m^2$ and integer n we define

$$v + G = \{v + u\colon u \in G\}.$$

A partition of the plane and the three-dimensional space into squares and cubes will be used so often that we introduce a special notation for it. For a positive integer P, we write

$$[P; i] = [iP\,..\,(i+1)P)$$

for the intervals of length P shifted by a multiple of P from the origin. Similarly, we write

$$[P; i, j]^2 = [P; i] \times [P; j]$$

for squares of size P shifted from the origin horizontally and vertically by a multiple of P. The cubes $[P; h, i, j]^3$ are correspondingly defined. The intervals of type $[P; i]$, $[P; i, j]^2$ and $[P; h, i, j]^3$ will be called P-intervals, P-squares, and P-cubes, respectively.

7.2. Configurations and Evolutions

Let S be a finite set, the set of all possible *states* of our cells. Let B be a set of sites. A *configuration* x over B is a function that orders a state $x[p] \in S$ to each element p of B. Let x, y be two configurations, over the sets of sites B and C, respectively. We will say that y is a *translation* of x, if there is a vector u such that $C = u + B$, and for all p in B we have $y[p + u] = x[p]$.

Let x be a configuration over the set B of sites, and C a subset of B. Often, we will talk about the configuration $x[C]$ that we obtain by restricting x to C. However, if C' is obtained by translation from C and x' is obtained by the same translation from x then we consider $x'[C']$ equal to $x[C]$. One dimensional example: if $C = \{-1, 0, 1\}$ then $x[t, p + C]$ is the string

$$(x[t, p - 1],\ x[t, p],\ x[t, p + 1])$$

(indexed by $0, 1, 2$).

An *evolution* with set S of states over the set $\mathbf{W}$ of sites in the time interval I is a function x from $I \times \mathbf{W}$ to S. we will write $x[t, p]$ for the state of cell p at time t in the evolution x. The function $x[t, \cdot\,]$

is a configuration for each t. Therefore $x[t, C]$ is defined according to the notation above.

7.3. Media and Their Trajectories

We will use the convenient term *medium* for an array of cellular automata. The set of sites is not part of the definition of a medium since we will consider the behavior of the same medium over various sets of sites (in particular, for various values of m).

A *medium* Med will be defined by giving a finite subset G of $\mathbf{W}$ (the *interaction neighborhood* of the site 0), a finite set $S = S_{\mathsf{Med}}$ of states, and a *transition function* Med: $S^G \to S$. Thus, the transition function Med orders a value $\mathsf{Med}(y)$ in S to all configurations y over G. A medium with a particular set of sites (always a torus in this paper) will be called an *iterative array*. We will often refer to the number

$$|\mathsf{Med}| = \lceil \log_2 |S_{\mathsf{Med}}| \rceil$$

as the *cell capacity* of medium Med.

Some evolutions of an iterative array are called trajectories. These are the evolutions x for which the value $x[t+1, p]$ depends only on the values $x[t, p+p']$ for p' in the neighborhood G, in a way determined by the transition function. An evolution x is called a *trajectory* of the medium Med if for all (t, p) we have

$$x[t+1, p] = \mathsf{Med}(x[t, p+G]). \tag{7.1}$$

The simplest example of an interaction neighborhood is $G = \{-1, 0, 1\}$ in one dimension. There, the function $\mathsf{Med}(y)$ can be simply viewed as an operation of three arguments in S, i.e., the right-hand side of (7.1) can be simplified to $\mathsf{Med}(x[t, p-1], x[t, p], x[p+1])$.

In our context, the word "error" could denote two different concepts. To distinguish these we call them "faults" and "deviations," and reserve the word "error" to informal discussion. We say that a *fault* occurred in x at $(t+1, p)$ (with respect to Med) if (7.1) does not hold. If y is a trajectory then the event $x[v] \neq y[v]$ is a *deviation* (of the evolution x from y). Intuitively, a medium is reliable if it can keep the number of deviations small despite the occasional occurrence of faults.

EXAMPLE 7.1 [Toom's two-dimensional error-correcting medium]. This medium can be defined for any state-space. Let us define first the majority function $\mathrm{Maj}(x, y, z)$. If two of the three arguments coincide then their common value is the value of Maj, otherwise $\mathrm{Maj}(x, y, z) = x$. The interaction neighborhood is defined by $H = \{(0, 1), (-1, -1), (1, -1)\}$, and the transition function R by $R(x[H]) = \mathrm{Maj}(x[H])$. Rule R computes the majority among the states of the northern, southwestern, and southeastern neighbors. Any constant function is a trajectory of R. (There are also some nonconstant periodic trajectories.) Suppose that $x[0, p] = 0$ for all p. We have a fault in $x[t, p]$ if it is not the value obtained by voting from the triple $x[t - 1, p + H]$. We have a deviation if $x[t, p] \neq 0$. □

Let $(\xi[t, p]: t \in [0 \,..\, l), p \in \mathbf{W})$ be a system of random variables with a joint distribution. We say that ξ is a *ϱ-perturbation* of medium Med if for each subset B of $[0 \,..\, l) \times \mathbf{W}$ the probability that for *all* $v \in B$ a fault occurs in ξ at v is less than $\varrho^{|B|}$. (This condition is satisfied if the faults occur independently with probability ϱ, but it includes some other cases important in statistical physics.) We will say that ξ is a ϱ-perturbation of a trajectory y over the same space–time set if it is a ϱ-perturbation of Med with $\xi[0, p] = y[0, p]$ for all p in $\mathbf{W}$. Our goal is to find situations in which if the probability ϱ is small then the probability of deviations is also small: in other words, faults do not accumulate.

7.4. Coding and Simulation

By reliability we mean the possibility of simulating the computation of a deterministic medium by an error-prone medium. For this, of course, the simulation must use an error-correcting, redundant code.

In this chapter, we will want to encode the information content of a square array of cells into a larger square array. Let P be a positive integer, and let C be a P-square. The set S^C is the set of all configurations over C.

Let S_0 and S_1 be two alphabets, and P_0, P_1 two integers, with C_i being the corresponding squares. A (two-dimensional block-) *code*

$$\psi = (\psi_*, \psi^*)$$

from S_1 to S_0 with *blocksizes* P_0, P_1 is characterized by *source blocksize* P_1, and *target blocksize* P_0, *encoding function*

$$\psi_*\colon S_1^{C_1} \to S_0^{C_0}$$

and *decoding* function ψ^* where $\psi^*(\psi_*(x)) = x$. A code can be *extended* to configurations larger than those over a single square: over a union of aligned squares, by encoding resp. decoding each square separately. We have a *single-letter* code if $P_1 = 1$.

Let us be given two media Med_0 and Med_1, with state sets S_0, S_1, and a code ψ with blocksizes P_0, P_1, further the natural numbers T_0, T_1. We say that ψ is a *simulation* of Med_1 by Med_0 with parameters P_0, P_1, T_0, T_1 if for any multiples m_i of P_i, any trajectories x_i of Med_i over the spaces $\mathbf{W}_i = \mathbf{Z}_{m_i}$, the relation

$$x_0[0, \mathbf{W}_0] = \psi_*(x_1[0, \mathbf{W}_1])$$

implies

$$x_0[T_0, \mathbf{W}_0] = \psi_*(x_1[T_1, \mathbf{W}_1]).$$

The parameters P_0, P_1 are still called the blocksizes. The parameter T_1 is called the *source work period*, while T_0 is called the *target work period*. Thus, a simulation sets up a correspondence between the evolutions of two media. A simulation is *single-step* if $T_1 = 1$.

A medium is *universal* if it can simulate any other medium by a total single-letter single-step simulation. There are many possible universal media in two dimensions. Some of them (the ones built like universal Turning machines) are easy to program. Others (like Conway's Game of Life, or Margolus's Billiard Ball rule) are very simple to define. For the purpose of the following theorems, let Med_0 be a fixed medium.

7.5. The Result

Our goal is to find a "reliable" simulation of our fixed arbitrary medium Med_0 by a suitable medium M, as Med_0 computes on a certain set $\mathbf{Z}_n^2$ of sites for a certain number t of steps. The simulation γ used will depend on the sizes n, t of the computation and the error probability ε permitted in the result.

The medium M and the structure of the code γ is complicated to describe, and we leave the description to the proof. However, there are two reasons why it is not possible to "cheat" and hide the whole computation in the code.

- The code is independent of the computation (except for its size), and decoding is an inverse of the encoding. To begin a computation on the output, we do not have to decode and encode again.
- The code can be computed rapidly.

Let m denote the target blocksize of the code γ. We say that the trajectory y of M on $\mathbf{Z}_m^2$ is in the *range* of γ if we have $y[0, \mathbf{Z}_m^2] = \gamma_*(u)$ for some configuration u of Med_0 on $\mathbf{Z}_n^2$.

THEOREM 7.1. (MAIN THEOREM). *There is a medium M, a fault probability bound $\varrho > 0$ and for each n, t, ε with $L = \log(n^2 t/\varepsilon)$, a simulation γ with parameters n, m, T, T', such that the following holds.*

- *for any $h < [t/T']$, the probability of the event*

$$\gamma^*(\xi[hT, \mathbf{Z}_m^2]) = \gamma^*(y[hT, \mathbf{Z}_m^2])$$

  is at least $1 - \varepsilon$ for all trajectories y of M in the range of γ, and all ϱ-perturbations ξ of y.
- *The periods are (almost) logarithmically small: we have $P, T = O(2^{(\log L)^{1.6}})$. The redundancies can be estimated by $m = O(n + P)$, $T/T' < L^{2+o(1)}$.*
- *The code γ is computable in $O(T)$ steps on a suitable deterministic medium.*

The code given in the theorem implements every computation of the ideal fault-free medium Med_0 in the "physical" medium M in such a way that the probability of deviations remains under control. The space requirement n of the original computation is increased to m in the implementation. Hence according to the statement of the theorem, the space redundancy is a constant factor, except when the space n is too small to accommodate even one (logarithmic-size) block of the simulation.

8. ERROR-CORRECTING CODES

8.1. Burst Error Correction

It is not surprising that the theory of reliable computation makes use of the theory of error-correcting codes. (What is surprising is how much more is needed besides error-correcting codes.) Indeed, if information is not stored in encoded form then one step of computation will be enough to cause irreparable damage.

A one-dimensional binary code γ (i.e., a code from strings to strings) is said to *correct t errors* if for all strings u, v, if $\gamma_*(u)$ differs from v in at most t places then $\gamma^*(v) = u$.

EXAMPLE 8.1 [Repetition code]. Let $\gamma_*(u) = uuuuu$ over strings u of length k. Let the decoding function be defined over strings v of length $5k$. The decoding goes as follows: to determine the $(i + 1)$th symbol of $\gamma^*(v)$, we look at the symbols with indexes $5i, 5i + k, \ldots, 5i + 4k$ in v and take their majority. It is easy to see that this code γ corrects two errors. Repetition coding uses too much redundancy. There are better error-correcting codes. □

The kind of error correction we need is measured rather in the number of *error bursts* of a certain size corrected than in the number of errors corrected. We say that code γ, mapping binary strings of length Kn to binary strings of length Nn, *corrects t bursts of size n* if it corrects any pattern of errors covered by at most t intervals of the form $[n; i]$.

The present section sketches the proof of the following theorem.

THEOREM 8.1. *Suppose that n has the form $2 \cdot 3^r$. Then for all integers $N \leqslant 2^n$, $t \leqslant 2^{n-1}$ there is a code from strings of length $(N - 2t)\,n$ to strings of length Nn correcting t error bursts of size n. There is a universal one-dimensional cellular array performing the decoding and encoding for these codes, for all n, N, t, in space $O(Nn)$ and time $O(Nn \log Nn)$.*

This theorem is a straightforward application of the theory of algebraic codes. The details of the proof are not original and are not needed for the rest of the chapter. We advise the average reader to just use the theorem and skip the rest of the present section.

8.2. Shortened BCH Codes

The BCH codes are treated, e.g., in the textbook [Bl]. A (shortened) BCH code correcting t error bursts of size n has a space of symbols that is a Galois field $GF(2^n)$, and needs $2t$ symbols devoted to redundancy. We represent the field as the set of remainders with respect to an irreducible polynomial. To make things completely explicit, we use the fact, derivable from the theory of fields (see [La]), that the polynomial

$$y^{2\cdot 3^r} + y^{3^r} + 1$$

is irreducible over GF(2). We choose n to be an integer of the form $2 \cdot 3^r$, and use the above irreducible polynomial to represent the field $GF(2^n)$.

Let α be the element of $GF(2^n)$ represented by the polynomial y. Then the elements $1, \alpha, \alpha^2, \ldots, \alpha^{N-1}$ are all different. Our codewords are the vectors $c = (c_0, \ldots, c_{N-1})$ over $GF(2^n)$ with the property that

$$\sum_j c_j \alpha^{ij} = 0 \ (i = 1, 2, \ldots, 2t).$$

or in other words, the polynomials $c(x)$ of degree $N-1$ over $GF(2^n)$ with the property that $c(\alpha^i) = 0$ for $i = 1, \ldots, 2t$. Let

$$g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{2t}).$$

Then the *codewords* are the polynomials $c(x)$ of degree $< N$ over $GF(2^n)$ divisible by $g(x)$. Hence the *information strings* can be represented by polynomials of degree $N - 2t - 1$ over $GF(2^n)$, and encoding is multiplication by $g(x)$.

We need efficient decoding and encoding algorithms to avoid significant time delay. Moreover, the total space used by our algorithms can be at most constant times more than the amount of information processed. To achieve this, we adapted some well-known algebraic algorithms to our iterative array.

8.3. Algebraic Operations

All needed algebraic operations can be performed in $O(n \log n)$ time and in $O(n)$ space. It is likely that time can be brought down to $O(n)$ but we do not need this fact.

Addition can obviously be done in $O(n)$ time and space.

Fast multiplication needs the fast Fourier transform (over an appropriate smaller field). The latter can be done by an algorithm $F(n)$ in $f(n) = O(n)$ steps, within a one-dimensional cellular array of length $O(n)$, as follows. In stage 1 of $F(n)$, we separate the even and odd digits into the two halves of the array: this can be done in $O(n)$ steps. Now a recursive call of $F(n/2)$ simultaneously transforms both halves in $f(n/2)$ steps. (For the sake of the Fourier transform, we can pad n to a power of 2.) Finally, the even and odd bits are restored from the two halves using $O(n)$ steps. This shows $f(n) = O(n)$. The overhead (e.g., a firing-squad-type organization for timing) does not take up more than $O(n)$ space.

The $O(n)$ Fourier transform gives $O(n)$ polynomial multiplication (convolution), and division, which give $O(n)$ multiplication over GF(n). For division over the field, the Euclidean algorithm is needed. It is known that the Euclidean algorithm can be organized in $O(n \log n)$ serial operations (see [Bo]). The same algorithm can be implemented here, even with the $O(n)$ space requirement, in $O(n \log n)$ steps.

8.4. The Complexity of Encoding and Decoding

Computing the polynomial $g(x)$ can be done in $O(N)$ field operations and linear space, using fast Fourier transform over GF(2^n). Encoding is multiplication by $g(x)$, hence it can be done with fast Fourier transform at the same time and space cost.

The first step of decoding is the computation of the syndromes $S_i = c(\alpha^i)$. Computing the values of the polynomial c at $2t$ places is a well-known operation doable on a sequential machine in $O(N \log N)$ operations. It does not cause any problem to adapt the known algorithm to the requirement of linear space on our cellular array.

The next step of decoding uses the Euclidean algorithm over GF(2^n), for finding the error-locator polynomial and error-evaluator polynomial: see Chapter 7.7 of [Bl]. Therefore it can be done in $O(nN)$ space and $O(N \log N)$ field operations, hence in $O(nN \log nN)$ operations altogether.

This completes the outline of the proof of Theorem 8.1. □

9. OUTLINE OF AN ERROR-CORRECTING STRATEGY

9.1. Correcting a Sparse Set of Faults

The assumption that faults arise randomly and independently is a natural one but not easy to deal with technically. In practice, if the fault probability is small then usually the different but similar assumption is made that faults just arise rarely. This could mean, e.g., that in not too large domains of time and space, no two faults will ever occur. Under such assumptions, error correction becomes easier. One can hope to find a mechanism to correct one fault, provided no new faults occur during the correction process. It turns out that even this problem needs an elaborate solution, outlined in Sections 10 and 12.

A little more generally, we introduce two integer parameters U and r. We will say that the set of faults is (U, r)-sparse if in each U-cube in space–time, there are at most r faults. If r is small enough with respect to U then media will be constructed that resists a (U, r)-sparse set of faults.

The present section outlines how these contributions can help in fighting probabilistic faults.

9.2. Probabilistic Noise Bounds

Let us be given some probability distribution on all possible sets in the space–time $V = \mathbf{Z} \times \mathbf{Z}_{mU}^2$. This distribution gives rise to a random set $\mathscr{E}$. For a parameter p, we say that the distribution of $\mathscr{E}$ is *p-bounded* if for all k and all finite sets $A = \{v_1, \ldots, v_k\}$ of space–time points, we have

$$\text{Prob}(A \subset \mathscr{E}) \leqslant p^k.$$

Let us define a new space–time $V^* = \mathbf{Z} \times \mathbf{Z}_m^2$ whose points are the U-cubes of V. Point $v = (h, i, j)$ of V^* corresponds to the cube $v_* = [U; h, i, j]^3$ of V. If $C = [U; h, i, j]^3$ then we will write $C^* = (h, i, j)$. For each subset E of V, we define the set $E^*(r)$ in V^* as follows: v is in $E^*(r)$ if there are more than r elements of E in the U-cube v_*. Via this mapping, the random noise $\mathscr{E}$ gives rise to the random noise $\mathscr{E}^*(r)$ in the space V^*.

LEMMA 9.1 (NOISE BOUND). *Suppose*

$$p < U^{-3(r+1)} \tag{9.1}$$

and that the noise $\mathscr{E}$ is p-bounded. Then the noise $\mathscr{E}^*(r)$ is p^r-bounded.

Thus, the derived noise is bounded with a much smaller probability than the original one.

Proof. By the definition of p-boundedness, for any set D of k space–time points, the probability of $D \subset \mathscr{E}$ is at most p^k. We can therefore increase the probabilities of all such events by assuming that individual points belong to the noise independently with probability p. Let us make this assumption. The following statement follows then immediately.

LEMMA 9.2. *For any sequence $B_0, B_1, \ldots$ of disjoint U-cubes, the events $B_i^* \in \mathscr{E}^*$ are independent.*

It follows from this lemma that it is enough to prove for a single U-cube B that the probability of $B^* \in \mathscr{E}^*$ is less than p^r.

For any sequence of $r+1$ points in B the probability that they are all in $\mathscr{E}$ is p^{r+1}. The total number of such sequences is less than $U^{3(r+1)}$. Therefore the probability that $B^* \in \mathscr{E}^*$, i.e., that there is such a sequence in B is less than $U^{3(r+1)}pp^r$. Hence, inequality (9.1) implies the statement of the lemma. □

9.3. Increasing Reliability by Simulation

Suppose that we could solve the problem of reliable computation in the presence of a (U, r)-sparse set of faults. This could mean, in the most simple-minded sketch, that for all U, r and media Med_0 satisfying certain simple conditions, we have a simulation φ and a new medium M_1 such that the decoding φ^* maps configurations x of M_1 over U-squares $[U; i, j]^2$ in the space $\mathbf{W} = \mathbf{Z}_{mU}^2$ into states $x^*[i, j]$ of Med_0 in the space $\mathbf{W}^* = \mathbf{Z}_m^2$. (We suppressed the dependence on U, r in this notation which is local to the present subsection.) The success in eliminating a (U, r)-sparse set of faults from the evolution x would mean that the decoded evolution $x^*[h, i, j]$ is a trajectory. In Sections 10–12, we will indeed solve the problem of reliable computation in the presence of a (U, r)-sparse set of faults, though the results will have a somewhat more complicated form.

If the set E of faults in x is not (U,r)-sparse then there will probably be faults in the evolution of x^* at the points of the derived set $E^*(r)$. *If the faults were confined to the set* $E^*(r)$ then the problem of error correction would be essentially solved. Indeed, according to the above lemma, the random set $\mathscr{E}^*(r)$ is p^r-bounded. We recreated therefore the original situation of a medium Med_0 to be implemented, and probabilistic faults, with the only difference that the probability bound is now p^r instead of p. We could call the simulation φ a "reliability amplifier," or in short, an *amplifier*. Concatenating several amplifiers, we can get probability bounds p^{r_1}, $p^{r_1 r_2}$, etc. Soon the probability of faults becomes negligibly small.

9.3.1. Too Big Cells

Before being carried away with this plan let us take a closer look. Each time we apply an additional amplification the cell capacity (we defined it as the logarithm of the number of states) of the simulating medium could increase. The number of amplifiers depends on the need to decrease the fault probability, and this depends on the size of the original computation. But this means that the cell capacity of the simulating medium depends on the size of the computation to be carried out, which is not what we want.

Due to this problem, Sections 13 and 14 modify the construction of Section 12, replacing each cell of the simulating medium by a block of some universal medium Univ. These blocks will be called *colonies*, and an array of colonies will be called an *alliance*. In the simulation thus modified, alliances in Univ will simulate blocks in the simulated medium Med_0. The working time T of colonies will be different from their size P. In case the simulated medium is also Univ the simulated blocks will also be called colonies. The simulated colonies are called *big*, the simulating colonies are called *small*.

9.3.2. How to Restore Colony Structure

Once we introduced colonies, a new problem arises, as mentioned in Section 5.2: the simulating colonies will need some minimal *structure* to work according to their program: how will this structure be restored after the faults? We will postulate this restoration *axiomatically*. Evolutions satisfying these axioms will be called *self-correcting evolutions*.

9.3.3. How to Obtain Self-Correcting Evolutions

We are willing to restrict ourselves to self-correcting evolutions if, in case the simulated medium is Univ again, the following holds:

> All evolutions obtained by decoding from evolutions that are self-correcting with respect to small colonies will be again self-correcting, with respect to big colonies.

The simulation, φ, will be modified in Section 16 to have this property. This will complete the construction of amplifiers.

10. A PERIODICALLY VARYING MEDIUM RESISTING SPARSE NOISE

10.1. The Noise Condition

In the present section, we start the investigation of a special kind of error correction. The concepts developed here have some interest in their own right. However, they are justified in the present chapter as a building block of the proof of the main theorem.

The noise condition used in the present section depends on a *time period* U and a *fault bound* r. Let us say that an evolution x is a (U,r)-*trajectory* if in every U-*cube*, x has at most r faults. We say that a (U,r)-*trajectory* is a (U,r)-perturbation of a trajectory y if it coincides with y at time 0. Our goal here is, for an arbitrary deterministic medium Med_0, to find a simulation φ of Med_0 by a suitable medium M_0 that withstands (U,r)-perturbation. The simulation has source- and target-blocksizes Q' and Q and source- and target-workperiods U' and U. It is simulating the work of Med_0 on $\mathbf{Z}^2_{nQ'}$.

The medium M_0 and the simulation φ used will not depend on the size n. They *do* depend on the medium Med_0 and the constants Q, U, Q', U', r. Thus, the error correction is block-for-block, using the fact that there are only r faults by U-cube.

The target blocks of our simulation will be called *alliances*. Each alliance has the form $[Q; i, j]^2$, consisting thus of $Q \times Q$ cells that, at least under fault-free conditions, form a cooperating unit.

10.2. Periodically Varying Media

Let us temporarily relax the requirement of homogeneity for the medium M_0. We permit the transition rule M_0 to be *inhomogeneous*:

it can change periodically in space (in both directions) with period Q and in time with period U. In other words, M_0 at space–time point t, i, j will depend on $t \bmod U$, $i \bmod Q$, and $j \bmod Q$: in the equation of a trajectory instead of 7.1 at a point (t, u) with $u = (i, j)$ we have

$$x[t+1, u] = M_0(x[t, u+G],\ t \bmod U,\ i \bmod Q,\ j \bmod Q).$$

We imagine such a transition rule as a computer program telling each cell of the alliances in each step of the working period specifically, what local action to perform.

Let $m = nQ$, denote the output blocklength of the code φ, and let $\mathbf{W} = \mathbf{Z}_m^2$.

Before stating the theorem let us note that the parameters Q, U, Q', U' are not completely arbitrary. The size Q must be large enough for $18r$ errors to be correctable. We require $Q > 3Q'$ to represent easily nine neighbor Q'-squares within one alliance. This requirement could be eased considerably since we can choose a larger cell capacity for the medium M_0. The time period U must clearly be long enough to carry out the error-correcting simulation. This will involve some decoding, coding, and repetition. A constant factor could again be hidden here by choosing a larger cell capacity in the simulating medium. For convenience, we also require the time period to be divisible by the blocklength. This leads to the following assumption.

CONDITION 10.1 (SIZE).

$$Q \geqslant \max(3Q', Q' + 22r),$$

$$U \geqslant \lceil U'/Q' \rceil Q^2 \log Q,$$

$$Q \mid U.$$

There are many ways to satisfy these conditions. Given an arbitrary r and Q', U', we can choose Q greater than the right-hand side of the first inequality. Then we can choose U to be any multiple of Q greater than the right-hand side of the second inequality.

THEOREM 10.1. *Suppose that the Size Condition 10.1 holds. Then there is a periodically varying medium M_0 with periods U, Q and capacity $|M_0| = O(\log(Q+U))$, and a simulation φ with parameters*

Q, Q', U, U' such that the following holds. For all n, all trajectories y of M_0 in the range of φ over $\mathbf{Z}^2_{nQ}$, all (U, r)-perturbations x of y, and all nonnegative integers h we have

$$\varphi^*(x[hU, \mathbf{W}]) = \varphi^*(y(hU, \mathbf{W}]).$$

The rest of the present section is devoted to the proof this theorem.

10.3. The Construction

Error-correcting devices have a crucial feature: *continuity*. The property of continuity means that every elementary event of the computation (happening at one space–time point) influences only a small part of the result. The continuity property will be achieved as follows. We subdivide the alliance into a certain number of columns and the working period into the same number of stages. We establish that in stage i, only column i can change its information content. In this way, stage i influences only column i.

10.3.1. Variables

As we often do in the analysis of Turing machines, the state-set $S = S_{M_0}$ of the medium M_0 will be the Cartesian product of several sets: $S = S_1 \times \cdots \times S_k$. Thus, the value $x[h, i, j]$ is the collection of several values $Z_1[h, i, j], \ldots, Z_k[h, i, j]$. For a Turing machine, we would say that we divided the tape into k individual "tracks." We call $\log|S_i|$ the *width* of track i.

Borrowing the terminology of computer programming, we will refer to the value

$$Z_1[h, i, j] = Z_1(x)[h, i, j]$$

as the *value of variable* Z_1 at site (i, j) at time t (in the evolution x). The transition rule M_0 will therefore say how the individual variables at site (i, j) depend on those in the neighbor sites at time h.

The notion of tracks (variables) is another tool to achieve continuity. We agree that the information used by the decoding function φ^* is in the variable *InpMem*, i.e., the value $\varphi^*(x[t, A])$ for the alliance A depends only on $InpMem[t, A]$. Now the program can

move information across any cells (i, j): as long as it does not require to change the variable $InpMem[i, j]$ and no fault happens at (i, j), the variable will not be changed.

Actually, the continuity requirement applies only to the variables in *InpMem* and the ones used immediately for their updating, called *OutMem*. We extend the notion of *deviation* to tracks, i.e., variables. When we compare the evolution x with the trajectory y of the same medium, we will say that there is a deviation at space–time point (t, i, j) on track Z_1 if $Z_1(x)[t, i, j]$ differs from $Z_1(y)[t, i, j]$. Of course, if x deviates from y on any one track at the space–time point (t, i, j) then there is a deviation in the absolute sense at this point, i.e., $x[t, i, j]$ and $y[t, i, j]$ are different. But in general, we will be more interested in the deviations on particular tracks than in deviations at all.

Let us outline the major operations that the inhomogeneous medium M_0 performs in the U steps of the work period in the alliance and the neighbor alliances.

10.3.2. Decoding

Let b be the first integer of form $2 \cdot 3^k$ greater than both the cell capacity $|\mathsf{Med}_0|$ and $\log Q$. The Size Condition and Theorem 8.1 implies the existence of a code Algeb from binary strings of length bQ' to binary strings of length bQ that corrects $11r$ error bursts of length b. The simulated Q'-square of Med_0 will be encoded row-by-row by the code Algeb.

Now we can define the code φ. It takes a Q' by Q' configuration of Med_0. It encodes each row into a binary string of length bQ'. Then it applies the code Algeb to each row, and writes the result onto the *ImpMem* track of a row of length Q, writing b bits into one cell. (Thus, the *ImpMem* track must be at least b bits wide.) The other tracks are set to an arbitrary initial state. There will be $Q - Q'$ unused rows in the target square: we can ignore them. Decoding is the inverse. We take the *ImpMem* track, and apply the decoding function Algeb to each of its rows.

The first task of the program is *decoding*. Indeed, we do not know how to manipulate the information in encoded form.

The decoding process, as well as all other operations mentioned later, uses tracks different from *InpMem* or *OutMem*. We will not give names to all these other tracks: they have the collective name Workspace. The result of the decoding in each alliance is a

configuration of Med_0 over a $Q' \times Q'$ square. The original content of the *ImpMem* variables is not changed. The result of the decoding is stored on a Workspace track.

10.3.3. Input

Due to the Size Condition, we can store the result of decoding in a subsquare of size $Q/3$ of the alliance, in a part of the Workspace called the Simulator track. In this way, the Simulator track of an alliance could store the decoded information not only from the alliance itself but also from its eight neighbors, arranged in their original geometrical relation, in nine subsquares. This is necessary, since the state of the alliance after U steps will also depend on the neighbor alliances.

10.3.4. Computation

On the decoded contents of the original blocks of Med_0, the work of Med_0 is simulated step-for-step for Q' steps, in the Simulator track. This procedure *Compute*(t) is simple: the Workspace track of the cells of M_0 is programmed to behave like Med_0 for t steps.

10.3.5. Encoding, Output, Repetition

The square of size $Q/3$ on the Simulator track is encoded again.

Now, we resist the temptation to write back the encoded result into the *OutMem* track of the colony. While decoded, the information was vulnerable to even a single fault. The computation performed on it could spread the errors even farther. Therefore we cannot completely trust the result. We will use only a single column of it, column s, therefore this part of the program can be called the procedure *Output*(s). In this procedure, we write column s of the result back into column s of a new track called *OutMem*. The rest of the result can be discarded. The reason we have to use a new track is that the old value of *InpMem* is still needed. Only the last step of the program replaces *InpMem* with *OutMem*.

The part of the program defined until now can be summarized in the following procedure.

```
procedure CompColumn(s, t);
begin
  Decode;
  Input;
  Compute(t);
  Encode;
  Output(s);
end;
```

One more level of repetition is needed. The parameter t above will always be chosen smaller than Q'. Indeed, more steps of the computation would depend on alliances farther away than the immediate neighbors. Therefore the program part defined until now must be repeated U'/Q' times.

The program below also has some idle steps at the beginning. These are not important for other than technical convenience in the later proofs.

The whole program can now be written as follows. Let

$$N = \lfloor U'/Q' \rfloor.$$

```
idle 2Q steps;
for l := 1 to N + 1 do begin
  if l ⩽ N then t := Q' else t := U' − NQ'.
  for s := 1 to Q do
    CompColumn(s, t);
  ImpMem := OutMem;
end
```

Since the rule M_0 is allowed to be space–time dependent, the implementation of the above program in the form of a transition table does not cause any principal difficulty. We do not do it because it is tedious. It is clear from Theorem 8.1 that all five parts of *CompColumn* take $O(Q \log Q)$ steps, hence the whole program takes at most $O(Q^2(U'/Q') \log Q)$ steps. A constant factor in the running time can be eliminated by the unusual speedup trick, increasing the cell capacity and combining several steps into one.

10.4. Proof of Theorem 10.1

Let y be a trajectory of M_0 over $\mathbf{W}$ in the range of φ. Let x be a (U, r)-perturbation of y. We have to prove the relation

$$\varphi^*(x[hU, \mathbf{W}]) = \varphi^*(y[hU, \mathbf{W}])$$

for all h. We will actually prove a little more:

LEMMA 10.1. *Let C be an alliance. Then for all h, in each row of C, on the InpMem track there are at most $10r$ deviations of x from y.*

The theorem will follow since the decoding φ^*, which is essentially the decoding Algeb*, corrects $18r$ errors. □

Proof of Lemma 10.1. The lemma certainly holds for $h = 0$, since x and y coincide there. We will assume that it holds for h and prove it for $h + 1$. Let C be an alliance. From the inductive assumption, it follows that at time hU, there are at most $10r$ deviations on the *ImpMem* track in any row of any neighbor alliance of C at time hU.

The events happening in the evolution x during the time interval $[U; h]$ that can have any effect on the configuration in C at time $(h + 1)U$ can obviously be covered by nine U-cubes. According to the assumption that x is a (U, r)-trajectory, in these nine cubes there are at most $9r$ faults.

The inner part of the program given above consists of Q calls to the procedure *CompColumn*(s, t), and a last step copying *OutMem* to *InpMem*. We are going to show that only those columns s of *InpMem* will have deviations on the *InpMem* track at time $(h + 1)U$ for which either there was a fault during the sth call or a fault in column s at some other time. The number of these columns is at most $9r + r = 10r$.

It is enough to show that in the calls of the procedure *CompColumn*(s, t) when no faults happen, no deviation is created in the sth column of the *OutMem* track. Certainly no deviations will be created during these calls on the *ImpMem* track, since nothing will be written there. Therefore columns containing deviations on the *InpMem* track in any of the neighbor alliances of C come from two sources. $10r$ of these columns were inherited from the beginning hU of the work period. r others are created by faults. Therefore

at the beginning of a call of *CompColumn*(s, t), there are at most $11r$ columns per alliance containing deviations on the *InpMem* track. Since the code Algeb can correct $11r$ errors per alliance, the effect of these deviations is eliminated by the procedure *Decode* during a fault-free call of *CompColumn*(s, t). □

11. TOOM'S RULE

11.1. Shrinking Deviations

Let us review the way a finite amount of information can be remembered in a two-dimensional homogeneous medium. For a finite set S of states, Toom's Rule R is defined in Section 7.3. Why is Toom's Rule R likely to preserve a nearly constant initial configuration? Let us define the linear functions

$$l_1(\alpha, \beta) = -2\alpha + \beta, \qquad l_2(\alpha, \beta) = 2\alpha + \beta, \qquad l_3(\alpha, \beta) = -\beta.$$

For an arbitrary subset F of $\mathbf{Z}^2$ we define

$$m_j(F) = \sup_{v \in F} l_j(v).$$

We call $m_j(F)$ the *measurements of* F. For any real numbers a_1, a_2, a_3, let us define the triangle

$$I = L(a_1, a_2, a_3) = \{(x, y)\colon l_j(x, y) \leqslant a_j \qquad \text{for } j = 1, 2, 3\}.$$

The numbers a_j are the measurements $m_j(I)$.

The following assertion is easy to verify. Let c be some constant, and let y be the constant evolution, i.e., for which $y[t, u] = c$ for all times t and sites u.

LEMMA 11.1. *Suppose that x is a trajectory of R in which at time t, the set of deviations from the constant evolution y is enclosed into the triangle $L(a, b, c)$. Then at time $t + 1$, the same set is enclosed in $L(a - 1, b - 1, c - 1)$.*

It is this speed of shrinking, *independent of size* of the set of deviations that distinguishes Toom's Rule.

11.2. Triangles

11.2.1. Size and Separation

For later use in proofs concerning Toom's Rule, let us introduce some more geometrical concepts. We call

$$|I| = (a + b)/2 + c$$

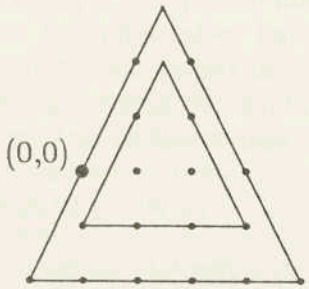
the size of triangle $L(a, b, c)$. [This expression must be chosen since the relation $(l_1 + l_2)/2 + l_3 = 0$ will then make sure that the size of a point is 0.] If the size is negative then the triangle is empty. For finite diameter m of the torus, the above definition does not work since inequality is not defined in $\mathbf{Z}_m$. But the definition can be easily extended as long as the size is less than m. Let us add this requirement to the definition of triangles.

For a set $\mathcal{J}$ of sets we denote by $\cup \mathcal{J}$ their union and by $|\mathcal{J}|$ the sum $\sum_{J \in \mathcal{J}} |J|$. We say that $\mathcal{J}$ *covers* a set if its union does so. We denote by $\# \mathcal{J}$ the number of elements of $\mathcal{J}$.

It is easy to verify the following: If the triangles I and J have nonempty intersection and $|I| + |J| < m$ (where m is the diameter of the torus) then the size of the smallest triangle containing their union is smaller than $|I| + |J|$. We can transform a finite set $\mathcal{J}$ of triangles with $|\mathcal{J}| < m$ into a set $\mathcal{J}'$ of disjoint triangles in the following way: we successively replace any pair of intersecting triangles in the set with the smallest triangle containing their union (this process will be called *merging*), as long as we find intersecting triangles. We have

$$|\mathcal{J}'| \leqslant |\mathcal{J}|.$$

Figure 1. Triangles $L(0, 6, 2)$ and $L(-1, 5, 1)$, of sizes 5 and 3, respectively.



For any sets H, I of the plane, let us define

$$d_j(H, I) = \inf\{l_j(x): x \in I\} - m_j(H).$$

This quantity is called the *j-separation* of the sets H and I. It is easy to check the following proposition, which can be considered the "separating hyperplane theorem" for triangles.

LEMMA 11.2. *Triangles H and I are disjoint if and only if there is a j such that their j-separation is positive. Moreover, we have*

$$d_j(H, I) = m_j(I) - \alpha_j|I| - m_j(H)$$

where $\alpha_j = 1$ for $j = 3$, and 2 otherwise.

11.2.2. Blowup and Deflation

For a triangle $I = L(a, b, c)$ and positive number d, we define the new triangle

$$D(I, d) = L(a - d, b - d, c - d)$$

called the *deflation* of I by the amount d. For a set $\mathcal{I}$ of triangles, we define

$$D(\mathcal{I}, d) = \{D(I, d): I \in \mathcal{I}\}. \tag{11.1}$$

For points $x = x_1, x_2$, $y = y_1, y_2$ we measure their distance by

$$\max(|x_1 - x_2|, |y_1 - y_2|).$$

For a set E in W and a positive number d we denote by $\Gamma(E, d)$ the set of points at a distance d or less from E. We call it the d-blowup of E. For a set E, let us denote by $D(E, -d)$ the smallest triangle I with the property that its deflation $D(E, d)$ contains E. These two operations are extended to sets similarly to (11.1). The relation between rectangular and triangular inflation is expressed by the following relations, which can be immediately verified. Let I be a triangle, p be a point not contained in I, and d a positive integer.

$$\begin{gathered}\Gamma(p, d) \subset D(p, -3d), \\ \Gamma(p, d) \cap D(I, 3d) = \varnothing.\end{gathered} \tag{11.2}$$

Both the deflation of triangles and the blowup of sets are additive, in the sense of the following property:

$$D(D(I, c), d) = D(I, c + d),$$

$$\Gamma(\Gamma(E, c), d) = \Gamma(E, c + d).$$

The first identity does not hold if $D(I, c)$ is empty but the right-hand side is not empty.

11.3. Reaching Consensus

Toom's Rule has the property that it preserves near consensus. Sometimes, we need a fault-resistant rule that besides this, *always establishes near consensus*, even from an arbitrary initial configuration. This problem is not unrelated to the group of problems known under the name "The Byzantine Generals Problem." In the present chapter, we will need this property in Section 16.

For the purposes of the present section, we will use a variant of the Toom Rule R defined in Section 7. In the rule R, a cell used the majority of the northern, southeastern, and southwestern neighbors for its next state. Now in the rule R' the cell uses the northern and eastern neighbor and itself. Rule R' is related to rule R but is simpler to analyze. (Indeed, it is easy to see that the effect of rule R is isomorphic to the combined effect of rule R' on four disjoint sublattices in space–time. We will not use this fact, but will just deal with the rule R'.)

We introduce an auxiliary rule *Inflate*. This rule replaces the value of a cell by the *disjunction* of the values of itself, its southern and western neighbors. Let us define a new rule in which the effect of Toom's Rule is biased by *Inflate*:

$$R^+ = \mathit{Inflate} \circ (R')^2.$$

Thus, each application of R^+ contains two applications R', followed by an application of *Inflate*. The interaction neighborhood of the new rule has, of course, diameter 3.

If x is an evolution then we will denote by $G_x(t)$ the set of sites u with $x[t, u] = 1$. The goal of the present section is to prove the following theorem. Let

$$\langle \mathrm{cons} \rangle = 32.$$

THEOREM 11.1. *Assume $m \geqslant 16$. Let $y[t, u]$ be a trajectory of R^+ over $\mathbf{Z}_m^2$. Then G_y ($\langle \text{cons} \rangle m$) is either empty or is the whole torus.*

The estimate $\langle \text{cons} \rangle m$ is certainly too high, but the present chapter cannot attempt to find the best coefficient of m in this theorem.

11.3.1. Geometrical Definitions

For the present subsection, triangles will be defined with the help of the linear functions

$$l_1'(\alpha, \beta) = -\alpha, \qquad l_2'(\alpha, \beta) = -\beta, \qquad l_3'(\alpha, \beta) = \alpha + \beta.$$

These are the triangles naturally associated with rule R' as the old triangles were with rule R. Now R' shrinks a triangle $L'(a, b, c)$ to $L'(a, b, c - 1)$. The *size* of a triangle $L'(a, b, c)$ is given by $a + b + c$. Thus, the rule decreases the size of each triangle by 1. We also introduce a new neighborhood relation on the two-dimensional lattice $\mathbf{Z}_m^2$. The new neighbors of a cell 0 are all vectors (i, j) with

$$\max(|i|, |j|, |i + j|) \leqslant 1.$$

(These are the old neighbors with the exception of the northeastern and southwestern ones.) We will view the set of sites as a graph where the edges connect the neighbors.

The mapping

$$(i, j) \to (i \bmod m, j \bmod m)$$

will be called the *wraparound*. For any cycle $u_1, u_2, \ldots, u_n$ in our graph we can compute the sum

$$(u_2 - u_1) + (u_3 - u_2) + \cdots + (u_1 - u_n)$$

not taken mod m but as the sum of integer two-dimensional vectors in the plane (not the torus). This sum always has the form (mi, mj) for some i, j. When $i = j = 0$ we say that the cycle is *contractable*. A connected subset of the torus is called contractable if all cycles in it are contractable. A noncontractable cycle is called a *belt*. The following lemma is elementary.

LEMMA 11.3. *A connected subset C of $\mathbf{Z}_m^2$ contains no belt if and only if there is a set C' in $\mathbf{Z}^2$ whose wraparound is C, such that the wraparound has an inverse on C that maps neighbors to neighbors. The set C' is unique up to translation.*

For a connected contractable set C we call the C' of the above lemma the *lift* of C. The *size* of a contractable connected component is the size of the smallest triangle containing its lift.

11.3.2. Global Behaviour of Toom's Rule

It is sometimes easier to think of the rule R' as applied not to a configuration x over $\mathbf{Z}_m^2$ but to the set G_x. Thus, for any transition rule D with a state-space $\{0, 1\}$ and any set $E \subset \mathbf{Z}_m^2$, we say $D(E) = E'$ if D has a trajectory y such that $E = G_y(0)$, $E' = G_y(1)$.

LEMMA 11.4. *Let C be a set of sites.*

(*a*) *Suppose that $R'(C)$ is nonempty. Then C is connected if and only if $R'(C)$ is connected.*
(*b*) *The set $R'(C)$ contains a belt if and only if C does. Suppose that C is connected, contractable with size $n > 0$. Then the size of $R'(C)$ is $n - 1$.*
(*c*) *If $C = C_1 \cup \cdots \cup C_n$ is the breakup of C into disjoint connected components then $R'(C) = R'(C_1) \cup \cdots \cup R'(C_n)$ is the breakup of $R'(C)$ into disjoint connected components (some of whom may be empty).*

The verification of this lemma is not difficult but requires the somewhat tedious examination of a few special cases, so we do not give it here.

THEOREM 11.2. *Let $y[t, u]$ be a trajectory of R' over $\mathbf{Z}_m^2$. We have $G_y(t) = 0$ for a large enough t if and only if $G_y(0)$ contains no belt.*

Proof. It follows from Lemma 11.4 (*b*) that if there is a belt in $G_y(0)$ then $G_y(t)$ never becomes empty. Suppose there are no belts $G_y(0)$. Then it follows from (*c*) and (*b*) of Lemma 11.4 that its components disappear after a finite number of steps. □

11.3.3. Combining Inflation with Toom's Rule

The following lemma is easy to check.

LEMMA 11.5. *Let $C = C_1 \cup \cdots \cup C_n$ be the breakup of C into disjoint connected components. Then each of the connected components of Inflate(C) is the union of sets of the form Inflate(C_i).*

The following inequality for an arbitrary set of sites follows from the monotonicity of the rule R'.

$$\mathit{Inflate}(R'(C)) \subset R'(\mathit{Inflate}(C)). \tag{11.3}$$

The following lemma follows easily from Lemmas 11.4 and 11.5.

LEMMA 11.6. *(a) Let $C = C \cup \cdots \cup C_n$ be the breakup of C into disjoint connected components. Then each of the connected components of $R^+(C)$ is the union of sets of the form $R^+(C_i)$.*
(b) If C is connected and $R^+(C)$ is contractable with size n then C is contractable with size $n+1$.

Let y be a trajectory of R^+. A connected component C' of $G_y(t+1)$ is said to be an *immediate successor* of a component C of $G_y(t)$ if $R^+(C)$ is nonempty and is contained in C'. A component D of $G_y(t+k)$ is a successor of a component C of $G_y(t)$ if there is a sequence $C_0 = C, C_1, \ldots, C_k = D$ such that C_i is a component $G_y(t+i)$ and is an immediate successor of C_{i-1}. The above lemma says that each component has at most one successor at all times.

LEMMA 11.7. *Let y be a trajectory of the rule R^+, let $n < m$. Let C be a connected contractable component of $G_y(t)$ which has only a single predecessor C_i at time i for all i in $(t-2n\,..\,t)$. Then C_{t-n} contains a triangle of size n.*

Proof. Since C is nonempty the predecessor at time $t-1$ has size at least 2. It follows from Lemma 11.6 that if C_{t-2n+1} is contractable then its size is at least $2n$. Let us denote temporarily $t-2n+1 = s$. We have, using (11.3):

$$\begin{aligned} C_{t-n+1} &= (R^+)^n(C_s) = (\mathit{Inflate} \circ (R')^2)^n(C_s) \\ &\supset \mathit{Inflate}^n((R')^{2n}(C_s)). \end{aligned} \tag{11.4}$$

Since C_s either contains a belt or has a size at least $2n$, Lemma 11.4 implies that the set $(R')^{2n}(C_s)$ is not empty. Therefore the right-hand side of (11.4) contains a triangle of size n. □

Proof of Theorem 11.1. Let y be a trajectory of R^+. Let us assume that $G_y(\langle \text{cons} \rangle m)$ is not empty. Let $t_0 = \langle \text{cons} \rangle m$. For $i = 0, 1, \ldots,$ let us define

$$\delta_i = \lceil \sqrt{48}(0.6)^{i/2} m \rceil, \qquad t_{i+1} = t_i - \delta_i.$$

First we show $t_i > 0$ for all $i \leqslant 4 \log m$. Indeed, we have

$$\begin{aligned} t_i &= t_0 - (\delta_0 + \cdots + \delta_{i-1}) \\ &\geqslant t_0 - i - \frac{\sqrt{48} m}{1 - \sqrt{0.6}} \\ &> \langle \text{cons} \rangle m - i - 31m \geqslant m - 4 \log m \geqslant 0. \end{aligned}$$

In the last two inequalities we used the definition of $\langle \text{cons} \rangle$ and $m \geqslant 16$.

If some $G_y(t_i)$ with $i > 0$, $t_i \geqslant 0$ contains a belt then $G_y(t_1)$ contains a belt. In the $\delta_0 \geqslant \sqrt{48} m$ steps until t_0 the component of this belt will be inflated over the whole space. Suppose therefore that $G_y(t_i)$ with $i > 0$, $t_i \geqslant 0$ has no belts.

Let n_i be the number of components in $G_y(t_i)$. Let us call a connected component in $G_y(t_i)$ *persistent* if it has exactly one predecessor in $G_y(t)$ for all t in $[t_{i+1} \,..\, t_i)$. Let j be the first i such that at least $n_i/6$ of the connected components in $G_y(t_i)$ are persistent.

First we show $j < 4 \log m$. For any $i < j$, at least $5n_i/6$ of the connected components are not persistent. Each of these components has at least two predecessors in $G_y(t_{i+1})$. We have therefore $n_{i+1} \geqslant 5n_i/3$ for all $i < j$. It follows that $n_i \geqslant (5/3)^i > 2^{i/2}$. For $i \geqslant 4 \log m$ this would give $n_i > m^2$, i.e., the number of components would exceed the size of the whole space.

Let C be a persistent connected component of $G_y(t_j)$. It follows from Lemma 11.7 that the predecessor of C in $G_y(t_j - \lceil \delta_j/2 \rceil)$ contains a triangle of size $\lceil \delta_j/2 \rceil$, i.e. it contains at least $\delta_j^2/8$ points. Altogether, the set $G_y(t_j - \lceil \delta_j/2 \rceil)$ contains thus at least as many points as

$$\frac{n_j}{6} \frac{\delta_j^2}{8} \geqslant \left(\frac{5}{3} \right)^j \frac{\delta_j^2}{48}.$$

Using the definition of δ_j, this is greater than number m^2 of points in the torus. □

12. A HOMOGENEOUS MEDIUM RESISTING SPARSE NOISE

12.1. Eliminating Space–Time Dependency

In this section, we construct a medium M_1 that does everything that M_0 does in Theorem 10.1, but it will be *homogeneous*, i.e., a medium in the original sense of our definitions.

For the sake of the present section, let us denote

$$\langle \text{sick} \rangle = 18$$

$$\langle \text{dev} \rangle = \langle \text{sick} \rangle + 18$$

$$\langle \text{corr} \rangle = 10\langle \text{sick} \rangle + \langle \text{dev} \rangle.$$

CONDITION 12.1 (SIZE).

$$Q \geqslant \max(3Q', Q' + 2\langle \text{corr} \rangle r),$$

$$U \geqslant ([U'/Q'])Q^2 \log Q,$$

$$Q \mid U.$$

There are many ways to satisfy these conditions, as shown in the remark after Condition 10.1.

THEOREM 12.1. *Suppose that the Size Condition 12.1 is satisfied. Then there is a medium M_1 and a simulation φ satisfying the assertions of Theorem 10.1.*

The present section is devoted to the proof of this theorem.

We can eliminate the inhomogeneity from M_0 *formally*, by introducing an extra restriction on the evolution. We add three new tracks, i.e., three new variables denoted by τ, π_1, π_2 called the *phase variables*. The variable τ takes its value from the set $[0\,..\,U)$. The variables π_1, π_2 take their values from $[0\,..\,Q)$. The new local state space is that of M_0, multiplied by the ranges of the phase variables.

We could restrict our attention to evolutions x over the new state-space with the property that at all space-time points (t, i, j) we have

$$\tau(x)[t, i, j] = t \bmod U,$$

$$\pi_1(x)[t, i, j] = i \bmod Q,$$

$$\pi_2(x)[t, i, j] = j \bmod Q. \tag{12.1}$$

This requirement means that even the faults cannot change the values of the phase variables. Now we can modify the transition rule of M_0 in such a way that instead of depending directly on time and space, it will depend on the phase variables. This new medium M_0' is homogeneous, and obviously satisfies Theorem 10.1, if we restrict ourselves to evolutions satisfying (12.1). In what follows we show how to eliminate the requirement (12.1).

12.1.1. Toom's Rule for Periodic Stable States

We will use Toom's Rule in a slightly generalized form. What we need to preserve are the periodic evolutions of the variables τ, π_1, π_2 rather than all-constant evolutions. (The generalized formulation of Toom's Rule is used in [Too] first.)

The rule for τ is (arithmetic operations are mod U):

$$\tau[h+1, i, j] = \mathrm{Maj}(\tau[h, i, j+1], \tau[h, i-1, j-1], \tau[h, i+1, j-1]) + 1.$$

The rule for π_1 is (arithmetic operations are mod Q)

$$\pi_1[h+1, i, j] = \mathrm{Maj}(\pi_1[h, i, j+1], \pi_1[h, i-1, j-1] + 1, \pi_1[h, i+1, j-1] - 1).$$

The rule for π_2 is analogous.

12.1.2. The Medium M_1 and the Code φ

Let us define the medium M_1 as follows. It works as the medium M_0' on the variables different from the phase variables. To the phase variables, it applies a generalized Toom's Rule as defined above.

The medium M_1 is homogeneous. We will prove that it satisfies Theorem 10.1, even if the property (12.1) is not required.

The code φ is defined similarly to its definition in the previous section. Decoding is exactly the same. In encoding, the phase variables must be set correctly. The phase variable τ will be set to 0. In this way, the result of the encoding is always an alliance at the beginning of its work period.

12.2. Legal Cells

Let x be an evolution. We will say that the cell at site (i, j) is *legal* at time t if the equations (12.1) are satisfied. Toom's Rule has the property that, in the absence of faults, it decreases the set of illegal cells. This property can be spelled out in a lemma similar to Lemma 11.1. In words, if the set of illegal cells is enclosed in a triangle then in the next moment, it will be enclosed in a smaller triangle. We generalize to sets of triangles. If there are several enclosing triangles then they will be able to contract onto the illegal cells only if each of them is surrounded by legal cells. For technical convenience, we will express this in the following form: there is a set of *disjoint* triangles whose *deflation* covers the illegal cells. Thus, we will use the following property of Toom's Rule.

LEMMA 12.1. *Let y be a trajectory of M_1 (not necessarily in the range of φ). Suppose that at time t all illegal cells are enclosed into $D(\mathcal{I}, 1)$, where $\mathcal{I}$ is a disjoint set of triangles. Then at time $t + 1$, all illegal cells are enclosed in $D(\mathcal{I}, 2)$.*

Let $m = nQ$, let y be a trajectory of M_1 over $\mathbf{W} = \mathbf{Z}_m^2$ in the range of φ. Let x be a (U, r)-perturbation of y.

Let E be an alliance. The set of sites at time $t - U$ that can have any effect on the state of E at time t is $\Gamma(E, U)$. Since squares of the order of magnitude U occur this way, we will try to use U-squares as much as possible. Condition 12.1 required U to be divisible by Q. Therefore each U-square is the union of some alliances. Let us call the U-squares therefore *alliance clusters*.

We will say that an alliance cluster C is *locally healthy* at time t if there is a set $\mathcal{I}$ of disjoint triangles with $|\mathcal{I}| \leqslant \langle \text{sick} \rangle r$ such that the deflation $D(\mathcal{I}, 1)$ covers the set of illegal cells in $\Gamma(C, U)$.

LEMMA 12.2. *Assume that the conditions of Theorem 12.1 hold. Let C be an alliance cluster. Then at all times hU, the following statements hold:*

(*a*) *In each alliance E of C, in each row of E, on the InpMem track there are at most* $\langle \text{dev} \rangle r$ *deviations of x from y.*
(*b*) *C is locally healthy.*

Of course, this lemma implies the theorem. □

To prove the lemma, we will use induction on h. It holds by definition for $h = 0$. Let us assume that it holds for h, we prove it for $h + 1$.

12.3. Singularity

First we will prove statement (*b*) of Lemma 12.2. Let us call a space–time point (t, i, j) *singular* if either a fault occurs in the evolution x at this point or (i, j) is illegal at time t. Otherwise, the point is called *regular*. At a regular space–time point (t, i, j), the medium M_1 computes the value $x[t + 1, i, j]$ just like the inhomogeneous medium M_0 did. Therefore our immediate goal is to obtain a bound on the set of singular points. The next lemma bounds their time projection by a small set of short intervals, and their space projection by a small set of triangles. First, some remarks and definitions.

Let C be an alliance cluster. The set of sites at the time $q \in [U; h]$ that can have any effect on the state of C at time $(h + 1)U$ is

$$C_q = \Gamma[C, (h + 1)U - q].$$

Let us define

$$\langle \text{kill} \rangle = (4.5\langle \text{sick} \rangle + 18)r + 1.$$

Notice that the first inequality in Size Condition 12.1 implies

$$10\langle \text{kill} \rangle < Q. \tag{12.2}$$

LEMMA 12.3 (SINGULARITY LOCALIZATION). *There is a set Y of times in* $[U; h]$ *consisting of* $9r$ *intervals of length* $\langle \text{kill} \rangle$, *and sets* $\mathscr{K}$, $\mathscr{L}$ *of triangles with*

$$|\mathscr{K}| \leqslant 9\langle \text{sick} \rangle r, \qquad |\mathscr{L}| \leqslant \langle \text{sick} \rangle r$$

such that the following holds. For any singular space–time points (q, i, j) *during* $[U; h]$ *with* $(i, j) \in C_q$ *we have*

$$q \in Y \cup [hU \,..\, hU + \langle \text{kill} \rangle), \qquad (i, j) \in \bigcup D((\mathscr{K} \cup \mathscr{L})', 1).$$

For $q > hU + \langle \text{kill} \rangle$, *the site* (i, j) *is covered even by the smaller set* $D(\mathscr{L}, 1)$.

This lemma implies statement (*b*) of Lemma 12.2. Indeed, inequality (12.2) implies $(h + 1)U > hU + \langle \text{kill} \rangle$. Therefore the set of singular sites, and thus certainly the set of illegal cells at time $(h + 1)U$ is covered by $D(\mathscr{L}, 1)$.

12.4. Triangle Development

12.4.1. Noise

The set C_{hU} consists of nine U-squares. The domain of space–time involved is covered by the nine cubes above these squares. Since x is a (U, r)-trajectory, there are at most $9r$ faults in these cubes. For a number q in $[U; h]$, let $\mathscr{F}_q$ be the set of projections of the faults in this domain happening at time q. The number q will be called *singular* if the set $\mathscr{F}_q$ is nonempty. Otherwise, it is called *regular*.

Let the set X consist of all singular numbers q. We define

$$Y = \bigcup_{q \in X} [q \,..\, q + \langle \text{kill} \rangle).$$

By definition, the set Y can be covered by $9r$ intervals of length $\langle \text{kill} \rangle$.

It is convenient to take the set $\mathscr{F}_q$ into account via the following set of triangles:

$$\mathscr{L}_q = D(\mathscr{F}_q, -1)'.$$

Here, each point of $\mathscr{F}_q$ was enclosed into a triangle of size 0 (itself), and then this triangle was blown up by 1. The blowup provides for the possibility of a later deflation by the same amount. The size of each element of $\mathscr{L}_q$ is 2. Therefore the sum of their sizes is at most $18r$.

12.4.2. The Triangle Sets $\mathscr{I}_q$

Let us define for all q in $[U; h]$ a set $\mathscr{I}_q$ of disjoint triangles such that the following proposition holds.

LEMMA 12.4. *At time q, the system $D(\mathscr{I}_q, 1)$ covers the illegal cells in C_q.*

The set $\mathscr{I}_q$ will be defined inductively. Since we assumed that C_{hU} is locally healthy at time hU, for each of the nine alliance clusters in C_{hU} there is a set $\mathscr{K}_i$ of triangles of size less than $\langle \text{sick} \rangle r$ for which $D(\mathscr{K}_i, 1)$ covers the sick cells at time hU. We define

$$\mathscr{I}_{hU} = \mathscr{K} = \left(\bigcup_i \mathscr{K}_i \right)'.$$

We proceed to the definition of $\mathscr{I}_q$ for $q > hU$. Let us assume that $\mathscr{I}_q$ is defined

$$\mathscr{I}_{q+1} = \begin{cases} (\mathscr{I}_q \cup \mathscr{L}_q)' & \text{if } q \text{ is singular,} \\ D(\mathscr{I}_q, 1) & \text{otherwise.} \end{cases}$$

With this definition, the proof of Lemma 12.4 is easy by repeated use of Lemma 12.1. □

12.4.3. Vanishing Triangles

The next lemma states that by the time $q = \langle \text{kill} \rangle$ the deflations eliminate the effect of the set $\mathscr{I}_{hU}$.

LEMMA 12.5. *For all $s \in [hU \,..\, (h+1)U - \langle \text{kill} \rangle)$ there is a q in $[s \,..\, s + \langle \text{kill} \rangle)$ for which $\mathscr{I}_q$ is empty. Consequently, for all $q \geqslant hU + \langle \text{kill} \rangle$ the triangles in $\mathscr{I}_q$ are covered by $D(\mathscr{L}, 1)$ with*

$$\mathscr{L} = \left(\bigcup_q \mathscr{L}_q \right)'.$$

The second statement follows because we built $\mathscr{I}_q$ by consecutive merging of elements of $\mathscr{L}_q$ to $\mathscr{I}_{hU}$.

Proof. Suppose that $\mathscr{I}_q$ never vanishes between s and $s + \langle \text{kill} \rangle$. Let us calculate the decrease of the size of $\mathscr{I}_q$ during this interval, combined with the sum of all increases since time hU. At singular points q, there is a possible increase by the size of $\mathscr{L}_q$. The sum of these increases is at most $18r$. There are at most $9r$ such points. For at least $\langle \text{kill} \rangle - 9r$ points in $[s \,..\, s + \langle \text{kill} \rangle)$, the decrease is at least 2. The total decrease from the size of $\mathscr{I}_{hU}$ thus is at least

$$2(\langle \text{kill} \rangle - 9r) - 18r = 2\langle \text{kill} \rangle - 36r > 9\langle \text{sick} \rangle r.$$

by the definition of $\langle \text{kill} \rangle$. But $9\langle \text{sick} \rangle r$ is the upper bound on the size of $\mathscr{I}_{hU}$. The size of $\mathscr{I}_q$ would thus have decreased below 0. The contradiction proves the lemma. □

Proof of Lemma 12.3. Let the space–time point (q, i, j) be singular. It follows from Lemma 12.4 and the definition of $\mathscr{I}_q$ that (i, j) is in a deflation of an element of $\mathscr{I}_q$, thus it is in $D((\mathscr{K} \cup \mathscr{L})', 1)$ since $(\mathscr{K} \cup \mathscr{L})'$ covers all sets $\mathscr{I}_q$. It follows from Lemma 12.5 that q is in $Y \cup [hU \,..\, hU + \langle \text{kill} \rangle)$, and that if $q > hU + \langle \text{kill} \rangle$ then (i, j) is in $D(\mathscr{L}, 1)$. □

12.5. Computation

To end the proof of the first statement of Lemma 12.2, let us remember that we assumed its assertion for h, and are proving it for $h + 1$.

Let us estimate the deviations at time $(h + 1)U$. Since we obtained an upper bound on the set of singular points during the space–time period of the computation, we can follow the proof of Theorem 10.1, i.e., the proof of Lemma 10.1. The program still consists of Q calls to the procedure *CompColumn*(s, t), and a last step copying *OutMem* to *InpMem*.

LEMMA 12.6. *If site (i, j) in column s has deviations on the InpMem track at time $(h + 1)U$ then either the duration of call s intersects the set Y or there was a singular event at (i, j) some later time.*

Proof. Only a singular event can create new deviations on the *InpMem* track. According to the Singularity Localization Lemma, the sites of the singular events are all covered by a deflation of the

set $(\mathscr{K} \cup \mathscr{L})'$. Therefore each row contains at most $|\mathscr{K}| + |\mathscr{L}| \leqslant 10\langle\text{sick}\rangle r$ new deviations on the *ImpMem* track. Each row contains at most $\langle\text{dev}\rangle r$ old deviations, making the total $(10\langle\text{sick}\rangle + \langle\text{dev}\rangle)r = \langle\text{corr}\rangle r$. Under the Size Condition 12.1, we can construct a code Algeb just as in Section 10.3, to correct this many deviations (error bursts, in the original terminology, but each "burst" is confined here to one cell). Therefore if a call s has no singular event its computation starts with the correct input and it will write the correct value on the *OutMem* track.

Lemma 12.3 says that the times of the singular events are covered by $Y \cup [hU \,..\, (hU + \langle\text{kill}\rangle))$. After the first idling steps of the program, the first call $s = 0$ begins at time $hU + 2Q$, which is, according to inequality (12.2), greater than $hU + \langle\text{kill}\rangle$. Therefore if a call s intersects with the time of a singular event then it intersects with the set Y. □

To finish the proof of the first statement of Lemma 12.2, we estimate the number of columns s for which either the duration of the sth call intersects the set Y or there was a singular event at some later time in the column.

The set Y consists of at most $9r$ intervals of length $\langle\text{kill}\rangle$. According to inequality (12.2) we have $\langle\text{kill}\rangle < Q$. Therefore each of these intervals can intersect the duration of at most two calls. This is at most $18r$ calls. The number of columns s in any row in which a singular event happened after iteration s can be estimated by $|\mathscr{L}| \leqslant \langle\text{sick}\rangle r$. The total number of deviations in any row at time $(h + 1)U$ is thus at most $18r + \langle\text{sick}\rangle r = \langle\text{dev}\rangle r$. □

12.6. Reaching Consensus in the Presence of Noise

This section applies the technique developed in the present section to the model introduced in Section 11.3. The notion of triangles, the constant $\langle\text{cons}\rangle$ etc. used here are therefore those used in Section 11.3.

THEOREM 12.2. *Let $x[t, u]$ be an evolution over $\mathbf{Z}^+ \times \mathbf{Z}_m^2$, and r an integer less than m. Suppose that x has at most r faults with respect to the rule R^+. Then there is an integer $b = 0$ or 1 such that for all $t > \langle\text{cons}\rangle(r + 1)m$ there is a set of triangles of total size less than $4r$ covering the set of sites u with $x[t, u] \neq b$. If at time 0, there is a set of triangles of total size less than $4r$ covering the set of sites u with $x[0, u] \neq b_0$ then $b = b_0$.*

The bound $\langle \text{cons} \rangle (r+1)m$ is probably too high. We conjecture that if $r = o(m)$ then $O(m)$ can be written in its place.

Notice that about m^2 processors are used here to fight r faults, where r is approximately m. This is analogous to the results in [Ly] where, in case of r *permanent* faults, approximately an r by r array of processors is needed to achieve consensus. (By permanent faults, we mean cells that can act arbitrarily.) The rule R^+ is sensitive to even a small number of permanent faults. Two permanent faults can keep an arbitrarily large triangle forever from shrinking. It is, an interesting open question whether, in case the consensus must be achieved by a homogeneous array of automata, an r by r array is necessary to achieve consensus in the presence of r transient faults. The rule R^+ can certainly be fooled by as many faults as the size of the torus, whether they are placed at one time, or at a constant number of places but for a long time. Is this true of all rules?

Sketch of proof. The proof of the second statement of the theorem is similar to (only much simpler than) the proof of the Singularity Localization Lemma 12.3. We build sets $\mathcal{I}_q$ of triangles and estimate their size increases and decreases.

To prove the first statement of the theorem, note that there is a time interval of length $\langle \text{cons} \rangle m$ between times 0 and $\langle \text{cons} \rangle (r+1)m$ when no faults occur. To this time interval, we can apply Theorem 11.1. □

13. COLONIES

In Theorem 12.1, we found a medium M_1 that, under certain noise conditions, reliably simulates a given medium Med_0. There is no universal medium for this sort of simulation, even if we fix the medium Med_0. Indeed, the cell capacity of the medium M_1 crucially depended on the pair (U, r). Nevertheless, we will bring all the different simulated and simulating media to a "common denominator." To standardize the simulated media, let us first choose some universal Turing machine Turing. For any medium Med_0, let us encode the alphabet S_{Med_0} into binary strings of fixed length $|\mathsf{Med}_0|$. This binary encoding is called *expanding*: we expand a symbol s into the string $\mathsf{bin}(s)$. Now let Prog_0 be some program (e.g., first one) that computes on Turing the transition function Med_0 from its nine arguments when each state s is represented by bin(s). The program Prog_0 describes the parameter $|\mathsf{Med}_0|$ as well.

The common denominator will be more interesting to find for the *simulating* media M_1 and different noise conditions (U, r). We define a new *universal medium* Univ. In the previous section, we simulated each Q'-square of Med_0 by a Q-square of M_1. Now for some integers P, T where T is a multiple of P, we subdivide the plane into squares of size P that will simulate, in a working period of size T, the cells of medium M_1. Just as we used a name for the Q-squares of medium M_1 calling them "alliances," we will use a name for the P-squares of medium Univ, calling them *colonies*.

Actually, we will dispose of the intermediate medium M_1 altogether. Instead of saying that we simulate M_1 by Univ and combine this with the simulation of Med_0 by M_1, we will just say that we simulate the Q'-squares of Med_0, using QP-squares of Univ called *alliances*. Here the medium Univ does not depend on Med_0 or any of the parameters.

13.1. Cells with Nonunit Size and Worktime

Later, we will consider evolutions of Univ that were obtained by decoding from some other evolution of another medium. In such cases, it is useful to measure the size and worktime of cells by the cost in space and time in the simulating medium. For this later goal, we generalize slightly the model considered so far. We introduce two, not necessarily integer, parameters, the *cell size* $\alpha \geqslant 1$ and *cell worktime* $\beta \geqslant 1$. If there are m cells across the torus then the space $\mathbf{W}$ will consist now not only of sites with integer coordinates but of all points with coordinates (i, j) where (i, j) are real numbers in $[0, m\alpha)$. Each cell of the space $\mathbf{W}$ occupies a square of size α. Similarly, each transition of the medium Univ will take β time units. Of course, the notions of evolution and trajectory are modified accordingly: evolution $x[t, u]$ is defined only for instants t that are multiples of β. Let us introduce the integers

$$P' = P/\alpha, \qquad T' = T/\beta.$$

During T units of time, only T' actual state transitions of the medium Univ occur, and the number of cells across a colony is only P'. It is reasonable to require that T' be significantly larger than P', in order for the colony to have time to receive information from the neighbor colonies and some time to process it. Let us require

$$T' \geqslant 9P'. \tag{13.1}$$

The numerical parameters determining the model are thus

$$P', T', Q, U, Q', U', P, T, r.$$

13.2. Noise

To the assumption that the medium M_1 has at most r faults in every U-cube, we make a corresponding assumption under the new conditions. The correspondence cannot be perfect since sites of M_1 correspond to squares of size P and working period T. Since we want to look at the colonies only at times $h = qT$, and since the effect of faults can be expected to spread during each work period, we account for faults by the abstract notion of *noise*. The noise is a given union $\mathcal{N}$ of some T-cubes.

Let us generalize the probabilistic noise bounds of Section 9. Let us be given some probability distribution on all unions of T-cubes. This distribution gives rise to a random union $\mathcal{E}$ of T-cubes. For a parameter p, we say that the distribution of $\mathcal{E}$ is *p-bounded* if for all k, all finite sets $A = C_1 \cup \cdots \cup C_k$ of disjoint T-cubes, we have

$$\text{Prob}(A \subset \mathcal{E}) \leqslant p^k.$$

The noise in a UT-cube C will be called (U, r, T)-*sparse* if at most r of the T-cubes of C belong to it. Since in the present section, U, r, T are fixed, we will not indicate the dependence on them. The noise is sparse over a union of UT-cubes if it is sparse on each of them. It is sparse over some other space–time set E if it is sparse over a union of cubes covering E.

Faults in M_1 were *transient*. After the fault happened, a cell of M_1 obeyed again the transition rule. Why would a fault in a colony have only a transient effect? The simulation that the colonies perform most probably depends on some structure within the colony that will be destroyed by the fault. Our present solution to this problem will be to *define it away*. We will make further *restrictions on the permissible evolutions*. The particular form of the restrictions is dictated by our needs to be able to *prove* them under certain circumstances.

Since T-squares $[T; i, j]^2$ will appear frequently below, we give them a name: we call them *clusters*. For a cluster E, we call the set $\Gamma(E, T)$ its *neighborhood*. It is the union of nine clusters.

13.3. Parameters and Phase Variables

Before we present the structure restoration condition we must say something about the notion of structure.

To find the boundaries of the colony at the beginning of the work period, let us mark the left-most column and the top row by a special symbol.

To give us some freedom for later tuning, let us add one more parameter: a number modif of yet unspecified role that will later be used to modify the action of Univ. Let us set aside the row second from the top for the parameters P', Prog_0, modif, r abbreviated as $P', \ldots, r$. This area is called the *parameter field.*

In the construction of the medium M_1 in the preceding section, three variables called the *phase variables* played a special role. The role of these variables will here also be a distinguished one. We set aside a fixed field for them in the colony called the *phase variable field:* the row below the parameter field.

Now we are ready to define a standard initial state for the colony. Let us assume that a special symbol set

$$S_{\text{init}} \subset S_{\text{Univ}}$$

and an element

$$topleft \in S_{\text{Univ}} \setminus S_{\text{init}}$$

are given. A colony $C = [P; i, j]^2$ is called *healthy* at time qT (with respect to the evolution x) if

- $x[qT, u] = topleft$ for all sites u that are either in the top row or in the leftmost column of C.
- $x[qT, u] \in S_{\text{init}}$ for all sites in C not in the top row or left-most column.
- The first two integers in the parameter field are P' and T'.

The notion of health thus depends on the parameters P', T'. This is not essential, but convenient.

A healthy colony is $[P; i, j]^2$ is *legal* with respect to the parameters $P', \ldots, r$ at time qT if the parameter field contains these parameters, and the phase variable contains the integers $q \bmod U$, $i \bmod Q$, and $j \bmod Q$.

The medium Univ *supports colonies* if the following condition holds, for all P', T' with $P' \mid T'$, for all trajectories y with unit cell sizes and worktimes and for all colonies C healthy at time 0.

- C is healthy at time T.
- The configuration of C at time T depends only on the configuration of its nine neighbors (including itself) at time 0.
- Assume that in addition, two of the northern, southeastern, and southwestern neighbor colonies are legal at time 0 with respect to some parameters $P', \ldots, r$. Then C is legal at time T.

We will be interested only in media Univ supporting colonies. Colony support will be easy to add to the program of a medium that does not have it.

Let φ be a simulation whose code maps Med_0-configurations $z[B]$ over a square B of size Q' into Univ-configurations $x[C] = \varphi_*(z[B])$ over a square C of size QP. We say that the simulation φ is *standard* if for each configuration $x[C]$ of the form $\varphi_*(z[B])$, all colonies in the alliance $x[C]$, are legal with the value 0 in the phase variable τ.

13.4. Damage, Quarantines

We introduce a way to keep track of the bad parts of a configuration $x[qT]$. For all q, we introduce a set $\mathsf{Damage}(qT) \subset \mathbf{W}$. This set consists not only of points that are sites (i.e., whose coordinates are multiples of α) but may contain also other points of the torus $\mathbf{W}$. For all clusters C the set $C \cap \mathsf{Damage}(qT)$ will be the union of a finite number of convex polygons. The set $\mathsf{Damage}(0)$ will be assumed given, and the set $\mathsf{Damage}(qT)$ will be defined inductively for $q > 0$.

It is often convenient to take the set $\mathsf{Damage}(qT)$ into account by a set of disjoint triangles containing it. Let C be a union of clusters. We will say that a set $\mathscr{I}$ of disjoint triangles is a *quarantine* at time qT on C if we have

$$C \cap \mathsf{Damage}(qT) \subset \bigcup D(\mathscr{I}, T).$$

The deflation $D(\mathscr{I}, T)$ is used in the above definition because the quarantine must cover the damage by a well-separated system of triangles.

For all C and qT there is a *minimal quarantine*. Indeed, the set $C \cap \mathsf{Damage}(qT)$ is the union of a finite set $\mathscr{F}$ of convex polygons. The set $D(\mathscr{F}, -T)'$ (see Section 11.2) is then the minimal quarantine.

Suppose that $\mathsf{Damage}(qT)$ is defined. Let C be a cluster. Let $\mathscr{I}$ be the minimal quarantine for $C' = \Gamma(C, T)$. We define

$$C \cap \mathsf{Damage}((q+1)T)$$
$$= \begin{cases} C \cap \bigcup D(\mathscr{I}, T+P) & \text{if } \mathscr{N} \cap ([T;q] \times C') = \varnothing, \\ C & \text{otherwise.} \end{cases}$$

Thus, the damage shrinks in the absence of noise, and it maximally grows in the presence of noise. We will be interested only in evolutions that satisfy the following condition.

CONDITION 13.1 (RESTORATION). *All colonies disjoint from* $\mathsf{Damage}(qT)$ *are legal at time* qT.;

This condition says, implicitly, that if, in the neighborhood of a cluster C, the illegal colonies are confined to well-separated triangles then, in the absence of noise, these triangles shrink.

We call a cluster $C = [T; i, j]^2$ *regular* at time qT if the following two conditions are satisfied.

- The neighborhood $C' = \Gamma(C, T)$ of C does not intersect with the noise during the interval $[T; q]$.
- $C' \cap \mathsf{Damage}(qT) = 0$.

Otherwise, we will call C *singular* at time qT. We will also call the triple (q, i, j) singular.

The following condition requires that the evolution of Univ we are considering should behave like a trajectory on regular clusters.

CONDITION 13.2 (COMPUTATION). *Let the cluster* C *be regular at time* qT. *Then the configuration* $x[(q+1)T, C]$ *is what it would be if* x *was a trajectory of* Univ *starting from the same configuration* $x[qT, C']$.

13.5. Summary of the Primitive Notions

We can divide the ingredients of the model into two parts. The first group contains those ingredients chosen by us, the machine

designers. The second group relates to the constraints imposed on the evolution that is otherwise chosen by nature.

Our choice:

- The parameters $P', T', Q, U, Q', U', P, T, \mathsf{Prog}_0, \mathsf{modif}, r$. (The notions of health and legality are now defined.)
- The colony-supporting medium Univ.
- The standard simulation φ.

Nature's choice:

- The evolution $x[t, i, j]$;
- The noise $\mathcal{N}$;
- The set Damage(0).

The triple $(x, \mathcal{N}, \mathsf{Damage}(0))$ will be called a *self-correcting evolution* if it satisfies the Restoration Condition and the Computation Condition. Thus, when we say that the evolution x is self-correcting, we assume that the other two ingredients of the triple are also given. The notion of a self-correcting evolution depends on the parameters $P', \ldots, r$ and the medium Univ.

A *self-correcting perturbation* of a trajectory y in the range of the simulation φ is a self-correcting evolution $(x, \mathcal{N}, \emptyset)$ such that x coincides with y at time 0. A random self-correcting perturbation of a trajectory y is a *self-correcting p-perturbation* if the random noise $\mathcal{N}$ is p-bounded. The problem of how to find random self-correcting perturbations with a small noise probability will be partly addressed in Section 13.7.

13.6. Size Conditions and Simulation Theorem

The theorem stated in the present section is analogous to the theorem of the previous section, using the Restoration Condition in place of Lemma 12.1. To emphasize the analogy and save notation, we will use some of the names and notation used in the preceding section with slightly different but analogous meaning. For the present section, let us define the constants

$$\begin{aligned} \langle \mathrm{sick} \rangle &= 72, \\ \langle \mathrm{dev} \rangle &= 180, \\ \langle \mathrm{corr} \rangle &= \langle \mathrm{dev} \rangle + 27(\langle \mathrm{sick} \rangle + 6). \end{aligned} \tag{13.2}$$

In the new Size Condition below the lower bound on QP' is needed for the existence of a code correcting enough error bursts and enough simulation space. The lower bound on P' makes sure that numbers smaller than U, Q and T as well as the program of the transition rule Med_0 are representable within a colony. The bound for UT' is similar to the one found in Size Condition 12.1, with UT' replacing U and $(QP/T)QT'$ replacing Q^2. The multiplier QP/T is the number of repetitions in the program. The expression QT' measures the time taken by Q colony work periods, i.e., the time to get any information across the alliance. The lower bound on T' is the same as (13.1). The divisibility assumptions serve only convenience. The constant Step_0 measures the time needed by the Turing machine Turing with program Prog_0 to compute the transition function Med_0. We assume that the size $|\mathsf{Prog}_0|$ of the program also measures the space needed for the computation.

CONDITION 13.3 (SIZE).

$$QP' > \max\left(3Q'|\mathsf{Med}_0|, Q'|\mathsf{Med}_0| + \frac{9\langle \mathrm{corr}\rangle rP'T}{P}\right), \quad (13.3)$$

$$P' > \max(\log U, \log T', |\mathsf{Prog}_0|, |\mathsf{modif}|), \quad (13.4)$$

$$UT' > \frac{U'}{Q'}\frac{QP}{T}QT' \log(QP')\mathsf{Step}_0, \quad (13.5)$$

$$T' > 9P', \quad (13.6)$$

$$P \mid T, \qquad \frac{T}{P} \,\Big|\, Q, \qquad Q \mid U.$$

Let us show how to satisfy all these conditions but (13.4). This remaining condition can be easily satisfied if we do not choose our parameters to be exponentially large compared to P'. Let Q', U', Prog_0 and modif be arbitrary. Let P' be large enough such that $|\mathsf{Prog}_0| < P'$. Let $P > P'$. Let $T > T'$ be a multiple of P large enough for (13.6). Let Q be a multiple of T/P large enough for (13.3). Let U be a multiple of Q large enough to satisfy (13.5). In this way, all Size Conditions will be satisfied.

THEOREM 13.1. *There is a medium* Univ *supporting colonies such that for all* $P', \dots, r$ *satisfying the Size Condition* 13.3 *with* modif $= 0$, *there is a standard simulation* φ *of* Med_0 *by* Univ *such that the following holds.*

Let y *be a trajectory of* Univ *in the range of* φ *over the space* $\mathbf{W} = \mathbf{Z}^2_{nQP}$, *and let* $(x, \mathcal{N}, \emptyset)$ *be a self-correcting perturbation of* y. *If the noise* $\mathcal{N}$ *is* (U, r, T)*-sparse then for all nonnegative integers* h *we have*

$$\varphi^*(x[hUT, \mathbf{W}]) = \varphi^*(y[hUT, \mathbf{W}]).$$

This theorem will be proved in the next section. The rest of the present section discusses the ways in which we will find self-correcting perturbations.

13.7. Implementations

We can view the medium Univ as the ideal, programmable medium to be used for computation. In order to achieve reliability in the real world, we have to find some physical "implementation" of Univ, in such a way that arbitrary evolutions of M are decoded into self-correcting evolutions of Univ.

The main ingredient of the implementation is a simulation ψ. Let the medium Univ support colonies. Let P', T', P, T be given satisfying

$$T' \geqslant 9P', \qquad P' \leqslant P, \qquad T' \leqslant T, \qquad P \mid T. \tag{13.7}$$

Let M be a medium and let ψ be a simulation with parameters P, P, T, T, that maps from the space $\mathbf{W}^*$ of Univ into the space $\mathbf{W}$ of M. (With the introduction of nonunit cell sizes, it is not restrictive to require that the simulating colonies in M have the same size and worktime as the simulated ones in Univ.) Let z be an evolution of M. We define the evolution $x = \psi^*(z)$ for values $t = qT$ as $x[qT] = \psi^*(z[qT])$, i.e., we obtain x by decoding from z. We define $x[t, u]$ for values t that are not a multiple of T in an arbitrary way.

Besides the decoding ψ^*, we must say how noise and the damage are found in the space $\mathbf{W}^*$. Let

$$\mathbf{\Psi} = (\psi, \mathcal{N}_{\mathbf{\Psi}}, \mathsf{Damage}_{\mathbf{\Psi}})$$

be a triple where ψ is the code assumed above. To each evolution, z of M, the mapping $\mathcal{N}_\Psi(z)$ orders a union of T-cubes in the space–time over $\mathbf{W}^*$. The mapping $\mathsf{Damage}_\Psi(z)(0)$ orders a subset of $\mathbf{W}^*$ to each evolution z.

Let $0 < p_1, p_2 < 1$. We call the triple Ψ an *implementation* with parameters

$$P', T', P, T, p_1, p_2$$

If the following properties hold.

- Suppose that the parameters $Q, \ldots, r$ supplement P', T', P, T in a way satisfying the Size Conditions. Then for all evolutions z of M, the triple

$$(\psi^*(z), \mathcal{N}_\Psi(z), \mathsf{Damage}_\Psi(z)(0))$$

  is a self-correcting evolution.
- Suppose further that for some trajectory y of Univ, the random evolution ζ is a p_1-perturbation of $\psi_*(y)$ (in the sense of Section 7). Then the triple $[\psi^*(\zeta), \mathcal{N}_\Psi(\zeta), \mathsf{Damage}_\Psi(\zeta)]$ is a self-correcting p_2-perturbation of y.

Of course, the notion of implementation depends on the choice of Univ. Our goal is to find implementations with constant p_1 and small p_2. This will be achieved in the following way. First, we find a trivial implementation in the paragraph below. Then, in Section 16, we show how to turn an implementation into one with a smaller p_2, using a special self-simulation of Univ that we will call an *amplifier*.

Let us give a trivial but important example Ψ_0 of an implementation.

LEMMA 13.1. *Let P', T', P, T be parameters satisfying* (13.7), *and $0 < \varrho < 1$. There is an implementation with the parameters $P', T', P, T, \varrho, \varrho$.*

Proof. We introduce a new medium M over the state set $S_M = S_{\mathsf{Univ}}^{P'^2}$ whose states are the healthy configurations over clusters $[T; i, j]^2$. The transition rule computes from the states of nine neighbor cells in one step what would have been computed in T'

steps by Univ from the corresponding clusters. The encoding ψ_{0*} of the code ψ_0 are equal to the identity function, defined on the healthy cluster configuration. In the space $\mathbf{W}$ of the medium M, we introduce cell size and cell workperiod T.

For an evolution z of M, a cube $[T; q, i, j]^3$ in $\mathbf{W}^*$ belongs to the noise $\mathcal{N}_{\Psi_0}(z)$ if the corresponding "cube" $[T; q, i, j]^3$ in the evolution z contains a fault. The set $\mathsf{Damage}_{\Psi_0}(z)(0)$ is the union of illegal clusters at time 0.

The Computation Condition will be satisfied automatically for an evolution x defined this way. Health is not a problem, since in this evolution, all colonies are healthy at all times, by definition. The Restoration Condition can be proved by induction on q since the medium Univ supports colonies. □

14. UNIVERSAL ROBUST SIMULATION WORKS

14.1. The Form of the Universal Simulation

The program of the simulation of an arbitrary medium Med_0 by colonies of Univ will be almost like the one for cells of M_1. In particular, the period-for-period application of the Toom Rule results in the colony-supporting property of Univ. Let us still point out some small differences.

14.1.1. The Error-Correcting Code

We represent each row of a Q'-square as a binary string of length $Q'|\mathsf{Med}_0|$. Let n be the greatest integer $\leqslant P'T/P$ of the form $2 \cdot 3^s$. We subdivide each row into segments of length n, using possibly a partial segment at the end: there are K segments, with

$$K = \lceil Q'|\mathsf{Med}_0|/n \rceil.$$

We use Theorem 8.1 to find an error-correcting code Algeb encoding these strings into strings of length

$$Nn = (K + 8\langle \mathrm{corr} \rangle r)n \leqslant Q'|\mathsf{Med}_0| + (8\langle \mathrm{corr} \rangle r + 1)P'T/P$$

and correcting $4\langle \mathrm{corr} \rangle r$ bursts of errors of length n. By Size Condition 13.3, these codewords are smaller than QP', therefore

will fit into a row of the alliance. The n-cell segments can reach across colony boundaries.

Now the code φ is defined much as in Sections 10 and 12. Decoding is the same. The encoding creates legal colonies that are about to start the working period of the alliance.

14.1.2. Variables into Fields

Some *variables* will now occupy several cells. This will happen, e.g., to the phase variables. Since the size of their field depends on Q, U, they cannot be part of the state of a single cell. We assigned to them the phase variable field.

14.1.3. Toom Phase Protection

To make colony support complete, we must apply Toom's Rule to the parameters and phase variables in every work period of every small colony. As long as a colony is healthy it does not cause any problem to do this. The parameters and phase variables of the northern, southeastern, and southwestern neighbor small colony are read in. After the application of the Toom's Rule to the parameters, the result is written into the parameter field. After this, using Q and U, the generalized Toom's Rule is applied to the phase variables. This operation is repeated in every T-interval, e.g., in parallel (using separate tracks) with everything else done by the colony.

14.1.4. Computation

Even after decoding by the code Algeb, the cell states of Med_0 are represented by binary strings of length $|\mathsf{Med}_0|$. The simulation part *Compute* of the program will be less direct now than it was when one cell of the simulated medium was represented in one cell of the simulating medium. The simulation of one step of Med_0 will be carried out for each cell of Med_0 with the help of the program Prog_0, simultaneously on all these strings.

14.1.5. Output Columns

Another slight difference between the new program and the old one is that in the old program, in the procedure *Output*(s, t), only

a single column was written. Now this procedure will write in a column of width T at once, i.e., in the part of the alliance whose projection is the interval $[T; s]$. This is the reason for the factor QP/T in the Size Conditions.

14.2. Local Health

Just as clusters (T-squares) are often more convenient than colonies, UT-squares are often more convenient than alliances. They will be called *alliance clusters*.

We will say that an alliance cluster C is *locally healthy* at time qT if it has a quarantine $\mathcal{I}$ with

$$|\mathcal{I}| < \langle \text{sick} \rangle rT.$$

For q in $[U; h]$, let

$$C_q = \Gamma(C, ((h+1)U - q)T). \tag{14.1}$$

This is the set of those sites whose state at time qT can have some effect on the state of the UT-square C at time $(h+1)U$. The next lemma is analogous to Lemma 12.2.

LEMMA 14.1. *Assume that the conditions of Theorem 13.1 hold, and the medium* Univ *and the code* φ *are defined as above. Let* C *be an alliance cluster. Suppose that at time* hUT*, the following statements hold:*

- *In each row of each alliance of* C_{hU}*, the deviations of the InpMem track of evolution* x *from the trajectory* y *are covered by* $\langle \text{dev} \rangle r$ *T-intervals.*
- C_{hU} *is locally healthy.*

If the noise is sparse over $C_{hU} \times [UT; h]$ *then the same statements hold at time* $(h+1)UT$.

The two statements of the lemma hold, of course, for $h = 0$. Therefore the lemma implies that they hold for all h, which will prove the statement of Theorem 13.1. □

14.3. Triangle Development

We begin with the lemma analogous to Lemma 12.3. Let

$$\langle \text{kill} \rangle = 11\langle \text{sick} \rangle rT/P.$$

The relation

$$10\langle \text{kill} \rangle < Q \tag{14.2}$$

follows immediately from the Size Conditions 13.3.

For any set $\mathscr{I}$ of triangles, let us denote

$$\mathscr{I}^0 = D(\mathscr{I}, T).$$

LEMMA 14.2 (SINGULARITY LOCALIZATION). *There is a set Y of times in $[U; h]$ consisting of at most $9r$ intervals of length $\langle \text{kill} \rangle$, and sets $\mathscr{K}$, $\mathscr{L}$ of triangles with*

$$|\mathscr{L}| \leqslant \langle \text{sick} \rangle rT, \qquad |\mathscr{K}| \leqslant 9\langle \text{sick} \rangle rT \tag{14.3}$$

with the following properties. Let $\mathscr{J} = (\mathscr{K} \cup \mathscr{L})'$. At all times qT with q in $[hU\,..\,(h+1)U]$, the set $\mathscr{J}$ is a quarantine for C_q. For $q > hU + \langle \text{kill} \rangle$, the same is true for $\mathscr{L}$.

Suppose that cluster E in C_q is singular at time qT for $q < (h+1)U$. Then q is in $Y \cup [hU\,..\,hU + \langle \text{kill} \rangle)$, and $\Gamma(E, T)$ is intersected by $\mathscr{J}^0$. If $q \geqslant hU + \langle \text{kill} \rangle$ then $\Gamma(E, T)$ is intersected by the smaller set $\mathscr{L}^0$.

This lemma implies the second statement of Lemma 14.1. Indeed, it says that the set $\mathscr{L}$, of size $\leqslant \langle \text{sick} \rangle rT$, is a quarantine for C at time $(h+1)U$.

14.3.1. Noise

The set $C_{hU} = \Gamma(C, UT)$ consists of nine alliance clusters. The domain of space–time involved is covered by the nine cubes above these squares. Since the noise is sparse, at most $9r$ of the T-cubes in this domain belong to the noise. For a number q in $[U; h]$, let $\mathscr{F}_q$ be the set of projections of the T-cubes belonging to the noise in this

domain at time qT. The number q will be called *singular* if the set $\mathscr{F}_q$ is nonempty. Otherwise, it is called *regular*. Let Y be the union of intervals $[q\,..\,q + \langle\text{kill}\rangle)$ for all singular numbers q.

To each T-square F in $\mathscr{F}_q$, we order a triangle $L = D(\Gamma(F, T), -T)$ with size $|L| = 8T$. Performing this operation for each element F of $\mathscr{F}_q$ we arrive at the set $\mathscr{L}_q$ of triangles. Let us define $\mathscr{L} = (\bigcup_q \mathscr{L}_q)'$. We have

$$|\mathscr{L}| \leqslant 9 \cdot 8rT = \langle\text{sick}\rangle rT.$$

14.3.2. The Triangle Sets $\mathscr{I}_q$

Let us define for all q a set $\mathscr{I}_q$ of disjoint triangles such that the following proposition holds.

LEMMA 14.3. *At time qT, the set $\mathscr{I}_q$ is a quarantine for C_q. If E is a cluster in C_q singular at time qT then $D(\mathscr{I}_q \cup \mathscr{I}_{q+1}, T)$ intersects $\Gamma(E, T)$.*

The set $\mathscr{I}_q$ will be defined inductively. Since we assumed that C_{hU} is locally healthy at time hUT, for each of the nine alliance clusters in it there is a quarantine $\mathscr{K}_i$ of size less than $\langle\text{sick}\rangle rT$. We define

$$\mathscr{I}_{hU} = \mathscr{K} = \left(\bigcup_{i=1}^{9} \mathscr{K}_i\right)'.$$

Then $\mathscr{K}$ is a quarantine for C_{hU}. We proceed to the definition of $\mathscr{I}_q$ for $q > hU$. Let us assume that $\mathscr{I}_q$ is defined. We define

$$\mathscr{I}_{q+1} = \begin{cases} (D(\mathscr{I}_q, P) \cup \mathscr{L}_q)' & \text{if } q \text{ is singular,} \\ D(\mathscr{I}_q, P) & \text{otherwise.} \end{cases}$$

We see that all sets $\mathscr{I}_q$ are covered by $\mathscr{I} = (\mathscr{K} \cup \mathscr{L})'$.

Proof of Lemma 14.3. This lemma is essentially an immediate consequence of the definitions.

Let us prove first that $\mathscr{I}_q$ is a quarantine for all q. We use induction on q. For $q = hU$, the statement follows from the definition of $\mathscr{I}_q$. Suppose it is true for q. If q is regular, it follows by the definition of $C_{q+1} \cap \mathsf{Damage}((q+1)T)$ given in 13.4.

If q is singular, it will also follow by this definition, applied to a smaller area B that is unaffected by the noise. We define

$$B = C_{q+1} \backslash \bigcup \Gamma(\mathscr{F}_q, T).$$

The set $\Gamma(B, T)$ is disjoint from the noise during the interval $[T; q]$. It follows from the damage definition that $D(\mathscr{I}_q, P)$ is a quarantine for B at time $(q+1)T$. We have now

$$\text{Damage}((q+1)T) \subset D(\mathscr{I}_q, P+T) \cup \Gamma(\mathscr{F}_q, T).$$

By the definition of $\mathscr{L}_q$, the second term on the right-hand side is contained in $D(\mathscr{L}_q, T)$. Using the definition of $\mathscr{I}_{q+1}$, this proves that it is a quarantine.

Suppose now that the cluster E in C_q is singular at time qT. Then one of the two conditions of regularity is not satisfied. If this is the first one then the set $\mathscr{I}_{q+1}$ contains $\mathscr{L}_q$ whose deflation contains E. If the second one is not satisfied then by the Restoration Condition the set $D(\mathscr{I}_q, T)$, as the deflation of a quarantine, intersects $\Gamma(E, T)$. □

14.3.3. Vanishing Triangles

Lemma 14.2 will follow from Lemma 14.3, the fact that $\mathscr{J}$ covers $\bigcup_q \mathscr{I}_q$, and the fact, to be proved in the present paragraph, that $\mathscr{L}$ covers $\bigcup_{q > \langle \text{kill} \rangle} \mathscr{I}_q$. □

LEMMA 14.4. *For all* $s \in [hU \,..\, (h+1)U - \langle \text{kill} \rangle)$ *there is a* q *in* $[s \,..\, s + \langle \text{kill} \rangle)$ *for which* $\mathscr{I}_q$ *is empty. Consequently, the triangles in* $\bigcup_{q > \langle \text{kill} \rangle} \mathscr{I}_q$ *are covered by* $\mathscr{L}$.

Proof. Suppose that $\mathscr{I}_q$ never vanishes between s and $s + \langle \text{kill} \rangle$. Let us calculate the decrease of the size of $\mathscr{I}_q$ during this interval, combined with the sum of all increases since time hU. At singular points q, there is a possible increase by the size of $\mathscr{L}_q$. The sum of these increases is at most $|\mathscr{L}|$.

There are at least $\langle \text{kill} \rangle - 9r$ regular numbers in the interval $[s \,..\, s + \langle \text{kill} \rangle)$. At each of these times, there is a deflation by P, i.e., a size decrease by $2P$. The total size decrease at these times is at least $2(\langle \text{kill} \rangle - 9r)P$. Combining with the possible increase of $|\mathscr{L}|$

and the original size $|\mathscr{K}|$, this gives

$$|\mathscr{I}_{s+\langle \text{kill} \rangle}| \leqslant |\mathscr{K}| + |\mathscr{L}| + 18rP - 2\langle \text{kill} \rangle P$$
$$< (10\langle \text{sick} \rangle + 18)rT - 2\langle \text{kill} \rangle P \leqslant 0.$$

By the definition of $\langle \text{kill} \rangle$, the size would thus have decreased below 0. This contradiction proves the lemma. □

14.4. Computation

Now we prove the first statement of Lemma 14.1 for $h + 1$. Let A be an alliance in the cluster C. Let us estimate the colonies with deviations at time $(h + 1)U$ in a row of A. Since we obtained an upper bound on the set of singular points during the space–time period of the computation, we can follow the proof of Theorem 10.1, i.e., the proof of Lemma 10.1. The program still consists of calls to the procedure *CompColumn*(s, t), for $s = 1, \ldots, QP/T$, and a last step copying *OutMem* to *InpMem*.

The following simple geometrical lemma is useful.

LEMMA 14.5. *Let $\mathscr{I}$ be a set of triangles. The number of clusters E in a horizontal row with the property that $\Gamma(E, T)$ intersects $\mathscr{I}^0$ is at most $3|\mathscr{I}|/T$.*

Proof. Let us estimate the number of elements of $\mathscr{I}$. We can assume that each element J of $\mathscr{I}$ has a size of at least $2T$, since otherwise, $D(J, T)$ would be empty. Therefore their number is at most $|\mathscr{I}|/(2T)$. It is easy to see that for a triangle J, the number of clusters E in a row for which $\Gamma(E, T)$ intersects J is at most $|J|/T + 4$. Therefore the total number of intersected T-intervals in a row is at most

$$\frac{|\mathscr{I}|}{T} + 4\frac{|\mathscr{I}|}{2T} = 3\frac{|\mathscr{I}|}{T}.$$ □

LEMMA 14.6. *If site (i, j) in column s will have deviation on the InpMem track at time $(h + 1)UT$, then either the duration of call s intersects the set Y or there was a singular event at (i, j) at some later time.*

Proof. Before time $(h + 1)UT$, only a singular event can create new deviations on the *InpMem* track. According to Lemma 14.2,

if cluster E is the site of a singular event then its neighborhood $\Gamma(E, T)$ intersects $\mathscr{J}^0$. According to Lemma 14.5, the number of such clusters in a row is at most $3|\mathscr{J}|$. Each row contains at most $\langle \text{dev} \rangle r$ T-intervals with old deviations, making the total

$$\langle \text{dev} \rangle r + 3|J|/T \leqslant (\langle \text{dev} \rangle + 27(\langle \text{sick} \rangle + 8))r = \langle \text{corr} \rangle r.$$

These T-intervals intersect at most four times this many n-intervals, according to the definition of the number n in the Section 14.1. The code Algeb was constructed to correct a pattern of $4\langle \text{corr} \rangle r$ bursts of errors of length n. Therefore if a call s has no singular event it will write the correct value on the *OutMem* track.

It follows from Lemma 14.2 that the singular numbers $q > hU + \langle \text{kill} \rangle$ are covered by the set Y. Since the call $s = 1$ begins only after $2QT'$ idling steps in the program and since we have inequality (14.2), the times of singular events during calls *CompColumn*(s) are covered by Y. □

Let us apply the above lemma to estimate the number of deviations at time $(h + 1)UT$. The set Y consists of $9r$ intervals of length $\langle \text{kill} \rangle$. By inequality (14.2), each of these intervals can intersect the duration of at most two calls. This is at most $18r$ calls. The number of intervals $[T; s]$ in any row in which a singular event happened after iteration s can be estimated similarly to the proof above by $3|\mathscr{L}|/T$. The total number of deviations in any row at time $(h + 1)U$ is thus at most.

$$3|\mathscr{L}|/T + 18r < 180r = \langle \text{dev} \rangle r.$$

This proves the first part of Lemma 14.1. The second part was proved after Lemma 14.4. □

14.5. The Spreading of Local Health

The second statement of Lemma 14.1 says that, under the conditions of Theorem 13.1, alliances always stay locally healthy. Let us strengthen this statement. A set $\mathscr{I}$ of triangles is a *local quarantine* for the alliance cluster C if there is a set $\mathscr{H}$ of triangles of size $\langle \text{sick} \rangle rT$ such that $(D(\mathscr{I}, UT) \cup \mathscr{H})'$ is a quarantine for C. We say that $\mathscr{I}$ is a local quarantine for a union of alliance clusters if it is a local quarantine for each of them.

The deflation by UT corresponds to the earlier deflation by T. Let us therefore define the abbreviation

$$\mathscr{I}^1 = D(\mathscr{I}, UT).$$

LEMMA 14.7. *Suppose that the medium* Univ *supports colonies and the parameters* $P', \ldots, r$ *satisfy the Size Conditions. Let* $(x, \mathscr{N}, \mathsf{Damage}(0))$ *be a self-correcting evolution. Let* C *be an alliance cluster,* h *a natural number, and* $\mathscr{I}$ *a local quarantine for* C_{hU} *at time* hUT. *Suppose that the noise* $\mathscr{N}$ *is sparse over the set* $C_{hU} \times [UT; h]$. *Then* $D(\mathscr{I}, QP)$ *is a local quarantine for* C *at time* $(h+1)UT$.

This lemma is a generalization of the second statement of Lemma 14.1 speaking about the preservation of local health. That statement becomes a special case with an empty set $\mathscr{I}$. The lemma is a step in our plan to *prove* the Restoration Condition for "higher order" colonies, simulated by alliances.

We will prove the above lemma by proving the following, somewhat stronger but less transparent, lemma which also contains a generalization for some parts of Lemma 14.2.

LEMMA 14.8. *Assume that the conditions of Lemma 14.7 hold. Let us define, for* $q \geqslant \langle \mathrm{kill} \rangle$,

$$\mathscr{H}_q = D(\mathscr{I}^1, (q - \langle \mathrm{kill} \rangle)P).$$

Assume that the noise is sparse over C_{hU} *during the time interval* $[UT; h]$. *Let the sets* $\mathscr{K}$, $\mathscr{L}$, Y *be defined as above. Then the set* $(\mathscr{I}^1 \cup \mathscr{K} \cup \mathscr{L})'$ *is a quarantine for* C_q *at time* qT *for all* q *in* $[hU \,..\, (h+1)U]$. *For* $q \geqslant \langle \mathrm{kill} \rangle$, *the same is true for the smaller set* $(\mathscr{H}_q \cup \mathscr{L})'$.

Suppose that cluster E *in* C_q *is singular at time* qT. *Then* $\Gamma(E, T)$ *is intersected by*

$$D((\mathscr{I}^1 \cup \mathscr{K} \cup \mathscr{L})', T).$$

If $q \geqslant \langle \mathrm{kill} \rangle$ *and* $\Gamma(E, T)$ *is not intersected by* $\mathscr{H}_q^0$ *then it is intersected by* $\mathscr{L}^0$ *and* q *is in* Y.

Lemma 14.7 follows immediately from Lemma 14.8. Indeed, the latter lemma implies that for all q, the set $D(\mathscr{I}, (q - \langle \text{kill} \rangle)P)$ is a local quarantine for C_q. Therefore for $q \geqslant \langle \text{kill} \rangle + Q$ the set $D(\mathscr{I}, PQ)$ is a local quarantine for C_q. It follows from (14.2) that $\langle \text{kill} \rangle + Q < 2Q < U$. □

Proof of Lemma 14.8. We build a new sequence of quarantines $\mathscr{I}_q$ just like in Section 14.3, satisfying Lemma 14.3. The recursive definition step is the same, but the starting quarantine $\mathscr{I}_{hU}$ is different. The set C_{hU} is the union of nine alliance clusters B_i for $i = 1, \ldots, 9$. Let $\mathscr{K}_i$ be the set of triangles with size $\langle \text{sick} \rangle rT$ such that $(\mathscr{I}^1 \cup \mathscr{K}_i)'$ is a quarantine for B_i. According to the assumption of the theorem, there is such a triangle set for all i. We define

$$\mathscr{K} = \left(\bigcup_i \mathscr{K}_i \right)', \qquad \mathscr{I}_{hU} = (\mathscr{I}^1 \cup \mathscr{K})'.$$

The latter set is obviously a quarantine for C_{hU} at time hU.

The process of creating $\mathscr{I}_q$ involves three kinds of steps. Sometimes we added a new triangle. Sometimes we deflated an old triangle. And sometimes we merged two triangles into the smallest one containing both. For each element of $\mathscr{I}_q$ we can define the set of their *ancestors*. New triangles are their own ancestors. Deflating a triangle does not change its ancestors. Merging two triangles unites their ancestry. Those triangles that have some ancestry in $\mathscr{I}$ are called *big*, the rest are called *small*.

LEMMA 14.9. *Each big triangle has exactly one big ancestor.*

Proof. We have to show that in the process of constructing $\mathscr{I}_q$, two big triangles will never be merged. Let H and I be two such triangles. They are disjoint, so according to Lemma 11.2, there is a j such that their j-separation $d_j(H, I)$ is positive. The construction begins by deflating each big triangle by the amount UT. This increases the j-separation by $2UT$. From that time on, the big triangles suffer only merging with small triangles and further deflation. Each deflation increases the separation. The merging decreases the separation at most by the size of the small triangle merged with the big one.

The small triangles come from elements of $\mathscr{K}$ or $\mathscr{L}_q$ for some q, using merging and deflation. Hence the small triangles are covered by $\mathscr{J}$ whose size is bounded by $10\langle\text{sick}\rangle rT$. The Size Conditions imply that this is smaller than $2UT$. □

Let $q > \langle\text{kill}\rangle$. We find just as in the proof of Lemma 14.4 that there is an s between hU and $\langle\text{kill}\rangle$ for which the small triangles disappear from $\mathscr{J}_s$. From this time on, the small triangles are covered by $\mathscr{L}$.

For each big triangle I in $\mathscr{J}$, let us denote by $\Lambda_q(I)$ the big triangle in $\mathscr{J}_q$ whose ancestor it is. Then we have

$$\Lambda_q(I) \subset D(I^1, P(q - \langle\text{kill}\rangle)). \tag{14.4}$$

Indeed, just as in the proof of Lemma 14.4, we find that the total extent of deflation of $\Lambda_q(I)$ is

$$\begin{aligned} &qP - 9rP - |\mathscr{J}| \\ &\quad \geqslant (q - \langle\text{kill}\rangle)P + \langle\text{kill}\rangle P - 10\langle\text{sick}\rangle rT - 9rP \\ &\quad \geqslant (q - \langle\text{kill}\rangle)P. \end{aligned}$$

Of course, we use Lemma 14.9 here.

This completes the proof of Lemma 14.8. Indeed, we showed that the big triangles and $\mathscr{L}$ together form a quarantine, and that the big triangles are deflated from $\mathscr{J}^1$ by at least $(q - \langle\text{kill}\rangle)P$. □

15. FORCING CODED OUTPUT

Section 14.5 indicated that the set of illegal colonies will shrink in the absence of noise. Thus, legality is restored automatically. Another important condition (used in the proof of Lemma 14.6) of successful computation in the presence of noise is that the input is close to a codeword of the error-correcting code Algeb. This condition will not be restored automatically, even if the colonies of the alliance are healthy, and even though the output of each fault-free computation is a codeword. Indeed, suppose that some input words have a larger number of deviations than the one correctable by Algeb. Then faults can change the input in such a way that the outputs of fault-free repetitions will be different from each other.

We keep only a small part of each repetition in *OutMem*. The result will therefore be a mixture of different codewords, which is in general not a codeword.

If the *InpMem* track contains words deviating only little from a codeword of Algeb then the result of the computation will be the same in all fault-free repetitions. In our program this implies that the result (even if it is otherwise wrong) will be close to a codeword.

The present section shows that a procedure can be added to the end of the program of the medium Univ that brings all words on the horizontal rows of the *InpMem* track of an alliance close to a codeword of Algeb, without significantly changing words that are already near codewords. This property will be important for the self-simulation in the next section.

For row $k = 0, \ldots, QP - 1$ of alliance E, let $u_k(q, E)$ denote the word on the *InpMem* track in the kth row of E at time qT. Let CodeDiff(u) denote the number of T-intervals in which the word u differs from $\mathsf{Algeb}_*(\mathsf{Algeb}^*(u))$. With an eye on the later application, we will only consider values $k > 2$, and will not try to change the top three rows of the alliance.

For the following lemma, two Size Conditions need some change with respect to Condition 13.3. Let

$$\langle \text{dev} \rangle' = 2\langle \text{corr} \rangle - \langle \text{dev} \rangle + 6\langle \text{sick} \rangle + 360.$$

We obtain $\langle \text{corr} \rangle'$ by replacing $\langle \text{dev} \rangle$ in its definition (13.2) with $\langle \text{dev} \rangle'$:

$$\langle \text{corr} \rangle' = \langle \text{dev} \rangle' + 27(\langle \text{sick} \rangle + 6).$$

The change in the Size Conditions is the following. In the first one, $\langle \text{corr} \rangle$ is replaced with $\langle \text{corr} \rangle'$. In the second one, QP/T is now squared.

CONDITION 15.1 (SIZE).

$$QP' > \max\left(3Q'|\mathsf{Med}_0|, Q'|\mathsf{Med}_0| + \frac{9\langle \text{corr} \rangle' rP'T}{P}\right),$$

$$UT' > \frac{U'}{Q'}\left(\frac{QP}{T}\right)^2 QT' \log(QP')\mathsf{Step}_0,$$

$$P' > \max(\log U, \log T', |\mathsf{Prog}_0|, |\mathsf{modif}|),$$

$$T' > 9P',$$

$$P \mid T, \qquad \frac{T}{P} \Big| Q, \qquad Q \mid U.$$

THEOREM 15.1. *There is a procedure ForceCode, with the following property. Suppose that the definition of the medium* Univ *and the code φ differs from the one given in Section 14 only in that ForceCode is attached to the end, and* Algeb *must be correct now $\langle \mathrm{dev} \rangle' r$ errors. Assume that the conditions of Theorem 13.1 hold, with the modifications in the Size Conditions indicated above.*

Let C be an alliance cluster. Suppose that at time hUT, the set C_{hU} is locally healthy. Let f_0 denote the beginning of the procedure ForceCode. Then in each alliance E of C in each row $k > 2$ of E, we have

$$\mathsf{CodeDiff}(u_k((h+1)U, E)) < \langle \mathrm{dev} \rangle r.$$

If we had $\mathsf{CodeDiff}(u_k(f_0, E)) < \langle \mathrm{dev} \rangle r$ *for all $k > 2$ then we have*

$$\mathsf{Algeb}^*(u_k(f_0, E)) = \mathsf{Algeb}^*(u_k((h+1)U, E))$$

for all $k > 2$. Thus, if all input words are close to a codeword the input will not be changed by ForceCode.

The rest of this section is devoted to the proof of the above theorem. The procedure *ForceCode* will attempt to decide whether each row of *InpMem* differs only slightly from a codeword of Algeb. If this is not the case then each row will be changed to a standard fixed codeword w_0.

The procedure works within each alliance independently. We will therefore omit the notation of dependence on E from $u_k(q, E)$. Whenever the time qT refers obviously to the current time we also omit the dependence on q.

In all of the following lemmas we assume that the number of T-cubes belonging to the noise in the neighborhood of the cluster of E in the given time period is at most $9r$, i.e., that the noise is (U, r)-sparse.

15.1. Computing the Votes

The crucial part of the procedure *ForceCode* writes a "vote" $Vote[i,s] = 0$ or 1 in each cluster (i,s). The following properties will hold, even in the presence of (U,r)-sparse noise.

- There is a number b and a set $\mathcal{I}$ of triangles of size $O(rT)$ such that $Vote[i,s] = b$ for all clusters (i,s) outside $\mathcal{I}$.
- If, at time f_0, for some $k > 2$, we have $\mathsf{CodeDiff}(u_k) > (2\langle \text{corr} \rangle - \langle \text{dev} \rangle)r$ then $b = 0$.
- If, at time f_0, for all $k > 2$, we have $\mathsf{CodeDiff}(u_k) \leqslant \langle \text{dev} \rangle r$ then $b = 1$.

The first part of *ForceCode* is an assessment of the situation. For the present section, let

$$n = PQ/T.$$

The

procedure *CheckCode* (i,s)

for $i,s = 0,\ldots,n-1$ does the following. After Q idling steps, it checks, simultaneously in all rows $k > 2$ of the alliance, whether

$$\mathsf{CodeDiff}(u_k) \leqslant \langle \text{corr} \rangle r.$$

Then it computes the conjunction of these tests and stores the result in a bit $Vote[i,s]$ in the cluster (i,s) of the alliance. Thus, only the place where the result is stored depends on i and s. The first part of *ForceCode* now looks like this.

```
procedure FindVotes
begin
  for i = 0 to n − 1 do
    for s = 0 to n − 1 do
      CheckCode(i, s);
end
```

The n^2 repetitions are performed as a guard against noise. This is the most time-consuming part of the whole program of Univ.

We assumed that C_{hU} is locally healthy. Let G_1 be the union of all clusters F for which $\Gamma(F, T)$ intersects a triangle in $\mathscr{L}^0$. They include all clusters in which a singular event occurred during the original program after the first Q idling steps. (By that time q, as shown in Lemma 14.8, the big triangles of the quarantine $\mathscr{I}$ of the definition of local health disappear in C_q.) Let G_2 be the union of all clusters (i, s) for which a singular event occurred during the nonidling part of the computation of *CheckCode* (i, s). The following lemma is obtained by an analysis of the above iteration similar to the proof of Lemma 14.6.

LEMMA 15.1. (*a*) *Suppose that at the beginning of the program* $\mathsf{CodeDiff}(u_k) \leqslant \langle \mathrm{dev} \rangle r$ *holds for all* $k > 2$. *Then at the end of FindVotes, in each cluster* (i, s) *not belonging to* $G_1 \cup G_2$ *we have* $Vote[i, s] = 1$.

(*b*) *Suppose that at the beginning of the program we have* $\mathsf{CodeDiff}(u_k) > \langle \mathrm{dev} \rangle r + 2(\langle \mathrm{corr} \rangle - \langle \mathrm{dev} \rangle) r$ *for at least one* $k > 2$. *Then at the end of FindVotes, in each cluster* (i, s) *not belonging to* $G_1 \cup G_2$ *we have* $Vote[i, s] = 0$.

15.2. The Consensus Problem

The real technical difficulty of the present section is that of making a decision based on the contents of the array $Vote[i, s]$. We cannot fight faults made during the decision as we did until now, by mixing the results from different repetitions. In borderline cases, the result could then namely be the mixture of some codeword and w_0. This mixture may not be a codeword, and we want a codeword result in all cases. We can view the values of $Vote[i, s]$ as the different opinions of some population. We want to achieve consensus in this population, without reversing a near consensus. This problem was essentially solved in Section 12.6.

To apply those results, the sites of a small torus (used there to achieve consensus) will be simulated by the clusters of the alliance. Now, a square is not a torus but a torus can be "folded over" to a square. To be more formal, let us introduce the mapping $i \to \tilde{i}$ from $\mathbf{Z}_{2n}$ to $[0 \,..\, n)$ by

$$\tilde{i} = \min(i, 2n - i - 1).$$

All points of the interval $[0..n)$ have two inverse images under this mapping. We define a mapping

$$(i, s) \to (\tilde{\imath}, \tilde{s})$$

of the torus $\mathbf{Z}_{2n}^2$ to a lattice square. This mapping maps points that are neighbors in the torus to points that are neighbors in the square. Let us temporarily denote by $(i_1, s_1), \ldots, (i_4, s_4)$ the four inverse images of (i, s).

We will use a new array $Vote'[i, s]$ over the torus $\mathbf{Z}_{2n}^2$. The four bits $Vote'[i_1, s_1], \ldots, Vote'[i_4, s_4]$ will be kept in the cluster $[T; i, s]^2$. Initially, the array *Vote* will be quadrupled into the array *Vote'*, by repeating each bit of a cluster four times. Of course, since one cluster represents four sites of the imaginary torus $\mathbf{Z}_{2n}^2$, one fault will affect all four of these sites. Now the second part of the procedure *ForceCode* can be written as follows.

procedure *Consensus*
begin
  for $i, s = 0$ **to** $2n - 1$ **parallelly do** $Vote'[i, s] := Vote[\tilde{\imath}, \tilde{s}]$;
  for $i = 1$ **to** $2(\langle \text{cons} \rangle + 1)n(r + 1)$ **do**
    $Vote' := R^+(Vote')$;
  for $i, s = 0$ **to** $n - 1$ **parallelly do**
    $Vote[i, s] := Vote'[i_1, s_1]$;
end

In the last step, we just chose arbitrarily the first one of the four votes in square $[T; i, s]^2$.

15.3. Analysis of the Consensus Algorithm

To avoid the awkward conversions between measurements, let us give a new size T to each cell of the torus $\mathbf{Z}_{2n}^2$, converting it into the real torus $[0, 2nT)^2 = [0, 2PQ)^2$ where the cells occupy the points whose coordinates are divisible by T (see Section 13.1). Without loss of generality, let us assume that the alliance E under consideration is $[PQ; 0, 0]^2$, i.e., its lower left corner is at the origin. The mapping $\tilde{a}$ now becomes $\min(a, 2PQ - a)$. In the torus $[0, 2PQ)^2$ we will also speak of T-squares. Each such T-square holds one vote.

In the reasoning below, we use both notions of triangles: the usual one, as well as the one used in connection with the modified Toom's Rule. Let us call the latter *special triangles*. The ordinary triangles are defined on the alliance cluster and its eight neighbors. The special triangles are defined on the torus $[0, 2PQ)^2$. Let us say that a special triangle *weakly covers* a cluster C if it covers the bottom of C.

LEMMA 15.2. *There is a number $b = 0$ or 1 such that at the end of the procedure Consensus, there is a set $\mathcal{M}$ of special triangles with*

$$|\mathcal{M}| \leqslant (6\langle \text{sick} \rangle + 360)rT$$

weakly covering each cluster $[T; i, s]^2$ of the torus with $Vote[i, s] \neq b$. In the cases treated by Lemma 15.1, the value of b will be what is stated there.

Proof. Let us first show that there is always a number b and a time t_0 during *Consensus* such that all votes outside $G_1 \cup G_2$ are equal to b. In the cases of Lemma 15.1 we can take t_0 to be the starting time of *Consensus*. In the rest of cases, let us locate a sequence of $2(\langle \text{cons} \rangle + 1)n$ noise-free iterations of the rule R^+. The first $2n$ iterations will be enough to eliminate the set $\mathcal{I}_q$, and thus make all clusters regular. By Theorem 11.1, the remaining $2\langle \text{cons} \rangle n$ noise-free iterations achieve consensus. Thus we can set t_0 equal to the time after these iterations.

The remainder of the proof will show that the consensus achieved by time t_0 will not be overturned. Assume $b = 0$. The case $b = 1$ can be treated similarly but somewhat simpler. We construct a set $\mathcal{M}$ of special triangles on the torus with the property that for all iterations of R^+ after t_0, they weakly cover every T-square in which the vote differs from 0.

For a triangle I, let us consider the four inverse images $I_1, \ldots, I_4$ of I under the above mapping $(i, s) \to (\tilde{\imath}, \tilde{s})$. Let $\bar{I}_k$ be the smallest special triangle containing I_k. For any set $\mathcal{I}$ of ordinary triangles let

$$\bar{\mathcal{I}} = \left(\bigcup_{I \in \mathcal{I}} \{\bar{I}_1, \ldots, \bar{I}_4\} \right)'.$$

Thus, we merge all special triangles obtained for all elements of $\mathscr{I}$ until we get a set $\bar{\mathscr{I}}$ of disjoint special triangles. We have $|\bar{I}_k| = 3/2|I|$, hence

$$|\bar{\mathscr{I}}| \leqslant 6|\mathscr{I}|.$$

Let us start from the set $\mathscr{L}$ of triangles. The number of triangles in $\mathscr{L}$ is at most $9r$. These have the property that for every cluster F that ever becomes singular during the computation considered, $\Gamma(F, T)$ intersects $\mathscr{L}$. The same will be true for $\bar{\mathscr{L}}$ and all inverse images of singular clusters. We have $\bar{\mathscr{L}} \leqslant 6|\mathscr{L}| \leqslant 6\langle\text{sick}\rangle rT$. Let us obtain the set $\mathscr{M}_1$ of special triangles by replacing each triangle $L(a, b, c)$ of $\bar{\mathscr{L}}$ with $L(a + 2T, b + 2T, c + 4T)$, and then merging. Then all inverse images of singular clusters are (completely) covered by $\mathscr{M}_1$. We have

$$\begin{aligned} |\mathscr{M}_1| &\leqslant |\bar{\mathscr{L}}| + 8T\#\bar{\mathscr{L}} \leqslant 6\langle\text{sick}\rangle rT + 8 \cdot 4 \cdot 9rT \\ &= (6\langle\text{sick}\rangle + 288)rT. \end{aligned}$$

If case (*a*) of Lemma 15.1 holds then at time t_0 set $G_1 \cup G_2$ can contain clusters (i, s) with $Vote[i, s] = 1$. We covered the inverse image of G_1 by $\mathscr{M}_1$. Let $\mathscr{M}_2$ be the set of special triangles obtained by covering the inverse images of the clusters of G_2 and merging. One cluster can be covered by a triangle of size $2T$. We have thus $|\mathscr{M}_2| \leqslant 4 \cdot 9r \cdot 2T = 72rT$. Let $\mathscr{M} = (\mathscr{M}_1 \cup \mathscr{M}_2)'$. The above estimates give $|\mathscr{M}| \leqslant (6\langle\text{sick}\rangle + 360)rT$.

Let us first show that if before an application of R', the set of clusters (i, s) on the torus with $Vote'[i, s] = 1$ is weakly covered by $\mathscr{M}$ then this is the case after the application, too. Suppose the cluster (i, s) (the T-square $[T; i, s]^2$) is not weakly covered by $\mathscr{M}$. Then it is regular. It is not possible that both clusters $[T; i, s + 1]^2$ and $[T; i + 1, s]^2$ are weakly covered by $\mathscr{M}$. Indeed, $\mathscr{M}$ consists of disjoint triangles, hence these two clusters would be weakly covered by the same triangle, which would then also cover the cluster $[T; i, s]^2$. Suppose, e.g., that $[T; i, s + 1]^2$ is not covered. Then the neighbor clusters $[T; i, s]^2$ and $[T; i, s + 1]^2$ are regular and contain a 0 vote throughout the computation of R'. Therefore the computation of R' will not change $Vote'[i, s]$.

Suppose that after an *Inflate*, at time $(q + 1)T$, a cluster $[T; i, s]^2$ gets vote 1. We will prove that it is weakly covered by $\mathscr{M}$. Suppose

it is not. Then it is regular. Suppose that one of the clusters $[T; i-1, s]^2$, $[T; i, s-1]^2$ is singular. Then it is completely (not only weakly) covered by a triangle of $\mathcal{M}$. It follows that $[T; i, s]^2$ is weakly covered by the same triangle. Suppose now that all three of these clusters are regular.

If *Inflate* brought $Vote'[i, s] = 1$ by time $(q+1)T$ then at time qT, we had either $Vote'[i-1, s] = 1$ or $Vote'[i, s-1] = 1$. Without loss of generality, let us suppose that we had $Vote'[i-1, s] = 1$. Since this is a regular cluster, this value of $Vote'$ at time qT is obtained by R' from the votes in clusters $[T; i-1, s]^2$, $[T; i, s]^2$, $[T; i-1, s+1]^2$ at time $(q-1)T$. Therefore two of these three clusters must be weakly covered by $\mathcal{M}$. It follows by simple geometric inspection that $[T; i, s]^2$ is also weakly covered. □

15.4. The New Codewords

For a word u and a number s in $[0\,..\,n)$ let us denote by $u[T; s]$ the part of u on the T-interval $[T; s]$. The next part

procedure *Enforce*

of *ForceCode* works simultaneously in all clusters of the alliance E. In cluster $[T; i, s]^2$, if $Vote[i, s] = 0$ then for all rows $k > 2$ in $[iT\,..\,(i+1)T)$, it changes $u_k[T; s]$ to $w_0[T; s]$.

LEMMA 15.3. *Let q denote the time at the end of the procedure Enforce. Under the conditions of Theorem 15.1, we have* $\mathsf{CodeDiff}[u_k(q, E)] \leqslant \langle \mathrm{dev} \rangle' r$ *for all $k > 2$. In case (a) of Lemma 15.1, we also have*

$$\mathsf{Algeb}^*(u_k(f_0, E)) = \mathsf{Algeb}^*(u_k(q, E)),$$

for all $k > 2$, i.e., the original codewords are not changed.

Proof. Suppose $\mathsf{CodeDiff}(u_k(hU, E))$ is at most $\langle \mathrm{dev} \rangle r$ for all $k \geqslant 2$. Then, by Lemma 15.1, we have the case $b = 1$ in Lemma 15.2. Due to the last lemma the clusters where the codeword is changed will be weakly covered by $\mathcal{M}$. Therefore the number of such clusters is bounded in each row by $|\mathcal{M}|/T$. This is the bound on the increase in the number of deviations from a codeword.

Suppose now that $\mathsf{CodeDiff}(u_k(hU, E))$ is greater than $\langle \text{dev} \rangle r + 2(\langle \text{corr} \rangle - \langle \text{dev} \rangle)r$ for some $k > 2$. Then by Lemma 15.1 we have the case $b = 0$ in Lemma 15.2, and we get $u_k = w_0$ in all clusters except for the ones weakly covered by $\mathcal{M}$. Therefore we get $\mathsf{CodeDiff}(u_k) \leqslant |\mathcal{M}|/T$.

Suppose finally that $\mathsf{CodeDiff}[u_k(hU, E)]$ is at most $\langle \text{dev} \rangle r + 2(\langle \text{corr} \rangle - \langle \text{dev} \rangle)r$ for all $k > 2$. Lemma 15.2 still holds but we do not know now the value of b. If $b = 0$ then the $\mathsf{CodeDiff}(u_k)$ will again be bounded by $|\mathcal{M}|/T$ for all $k > 2$, while if $b = 1$ then the number of deviations will *increase* by $|\mathcal{M}|/T$, bringing it to at most

$$\begin{aligned} &\langle \text{dev} \rangle r + 2(\langle \text{corr} \rangle - \langle \text{dev} \rangle)r + |\mathcal{M}|/T \\ &\quad = (2\langle \text{corr} \rangle - \langle \text{dev} \rangle + 6\langle \text{sick} \rangle + 360)r \\ &\quad = \langle \text{dev} \rangle' r. \end{aligned}$$

□

The final part of the procedure *ForceCode* is a procedure that will decrease $\mathsf{CodeDiff}(u_k)$ for each $k > 2$ from $\langle \text{dev} \rangle' r$ back to $\langle \text{dev} \rangle r$. It will simply try to decrease the difference to 0. To fight faults it is repeated for each output column of width T just as *CompColumn*(s, t).

```
procedure Refresh
begin
  for s = 0 to n − 1 do
    for k := 0 to PQ − 1 parallelly do
      u_k[T; s] := (Algeb_*(Algeb*(u_k)))[T; s]
end
```

End of proof of Theorem 15.1. The proof of Lemma 14.6 applied to the procedure *Refresh*, together with the fact that the code Algeb corrects $\langle \text{dev} \rangle' r$ bursts of errors, gives the desired conclusion. □

Let us summarize the procedure *ForceCode*.

```
procedure ForceCode
begin
  FindVotes;
  Consensus;
  Enforce;
  Refresh;
end.
```

16. AMPLIFIERS

Theorem 13.1 and Lemma 14.8 look similar to the Computation Condition and Restoration Condition for alliances instead of colonies. This section extends the similarity to equivalence. We will show that by an appropriate choice of the code φ, a new self-correcting evolution can be defined by decoding. With the tools of the present section, we learn how to turn an implementation into a better one, as desired in Section 13.7.

In the previous section, the medium Med_0 was chosen arbitrarily. Let us choose it now to be the medium Univ. There is no circularity in this choice: the medium Univ was constructed to work for *all* media Med_0, including Univ itself. Let us refer to the spaces in which the simulating and simulated iterative arrays of the medium Univ operate by $\mathbf{W}^*$ and $\mathbf{W}$, respectively:

$$\mathbf{W}^* = \mathbf{Z}^2_{nQ'}, \qquad \mathbf{W} = \mathbf{Z}^2_{nPQ}.$$

Let us assume that the parameters $P', \ldots, r$ satisfy the Size Condition 13.3. Let φ be a code satisfying the conditions of Theorem 13.1.

The decoding φ^* orders an evolution x^* of $\mathsf{Med}_0 = \mathsf{Univ}$ in $\mathbf{W}^*$ to the evolution x of Univ in $\mathbf{W}$. Let us define

$$y = \varphi_*(x^*). \tag{16.1}$$

Note that the evolution y is now not necessarily a trajectory. The target blocks of the code φ are the PQ-squares called alliances. The source blocks are Q'-squares, which we will call *big colonies*. We define

$$T^* = TU, \qquad P^* = PQ, \qquad \alpha^* = TU/U', \qquad \beta^* = PQ/Q'$$

Then we have $P^* \mid T^*$. We assign nonunit cell size α^* and cell worktime β^* to the simulated space. With this definition, the size and work time of the big colony is equal to the size and work time of the alliance simulating it. The total sizes of the spaces $\mathbf{W}$ and $\mathbf{W}^*$ become identical. Both spaces are subdivided into the same number of squares of size $P^* = PQ$. However, the size of the individual cells is generally larger in $\mathbf{W}^*$ than in $\mathbf{W}$.

Under the simulation φ, the evolution x^* on the big colony $E = [P^*; i, j]^2$ during time interval $[T^*; h]$ corresponds to the evolution

x on alliance $E_* = [QP; i, j]^2$ in the time interval $[UT; h]$. Just as T-squares were called clusters, the T^*-squares will be called *big clusters*. The above correspondence will map each big cluster A into an alliance cluster A_*. Therefore each T^*-cube $B = [T^*; h] \times A$ is mapped into a UT-cube $B_* = [UT; h] \times A_*$.

16.1. Noise, Health, Damage

We will say that the T^*-cube B belongs to the noise $\mathcal{N}^*$ if the noise $\mathcal{N}$ is *not sparse* on the corresponding UT-cube B_*. Notice that the noise $\mathcal{N}^*$ is defined in terms of the noise $\mathcal{N}$ and the sizes T, U, r, independently of all other primitive notions.

Let us choose a Q^* and $U^* > Q^*$ such that $\log U^* < Q'$. Now that we have P^*, T^*, Q^*, U^*, the notion of health and legality for big colonies will be defined just as it was for small colonies. The set Damage*(0) is the union of all big clusters C for which *not* all of following conditions hold at time 0:

- The alliance cluster C_* is locally healthy.
- In each row of each alliance of C_*, the deviations of the *InpMem* track of evolution x from the evolution y are covered by $\langle \text{dev} \rangle r$ T-intervals.
- All big colonies in C are legal.

The first two conditions concerns the simulating evolution x. They are identical, for $h = 0$, with the statements in Lemma 14.1. The second condition can also be stated, using the terminology of Section 15 as $\text{CodeDiff}(u_k(0, C_*)) \leqslant \langle \text{dev} \rangle r$ for all rows k of C_*. The last condition concerns the simulated evolution x^*.

16.2. Simulating a Self-Correcting Evolution

THEOREM 16.1. *There is a universal medium* Univ *supporting colonies with the following properties. Assume for parameters* $P', \ldots, r$, *the Size Condition 15.1.*

- *If* modif $= 0$ *then* Univ *satisfies Theorem 13.1.*
- *If* modif $= 1$ *then there is a standard simulation* φ *of* Univ *by itself such that for all self-correcting evolutions* $(x, \mathcal{N}, \text{Damage}(0))$ *the triple* $(x^*, \mathcal{N}^*, \text{Damage}^*(0))$ *is a self-correcting evolution.*

This theorem forms the backbone of the proof of the main result, Theorem 7.1. Its case with modif $= 0$ is simply Theorem 13.1. The case with modif $= 1$ gives the desired *amplifier* simulation that turns an implementation into a better one, with smaller probability bound. It implies namely the following theorem.

THEOREM 16.2. *Suppose that the parameters $P', \dots, r$ satisfy the conditions of Theorem 16.1 and the medium* Univ *satisfies its conclusion. Suppose that we have an implementation Ψ with the parameters P', T', P, T, ϱ, p, satisfying* (9.1). *Then the construction*

$$\Phi\colon (x, \mathcal{N}, \mathsf{Damage}(0)) \to (x^*, \mathcal{N}^*, \mathsf{Damage}^*(0))$$

gives a new implementation $\Psi \circ \Phi$ with the parameters $Q', U', P^*, T^*, \varrho, p^r$.

The rest of the section is devoted to the proof of Theorem 16.1, i.e., to the changes to the program of Univ to achieve the desired effects for modif $= 1$.

First we prove an auxiliary statement.

LEMMA 16.1. *Suppose that the medium* Univ *supports colonies and the parameters $P', \dots, r$ satisfy the Size Conditions 15.1. Let $(x, \mathcal{N}, \mathsf{Damage}(0))$ be a self-correcting evolution. Let C be a big cluster in W^*, and h a natural number. Then any quarantine at time hT^* for C with respect to the triple $(x^*, \mathcal{N}^*, \mathsf{Damage}^*(0))$ is a local quarantine for C_* with respect to the triple $(x, \mathcal{N}, \mathsf{Damage}(0))$.*

Proof. This statement can be proven by induction on h. It is true for $h = 0$ by the definition of $\mathsf{Damage}^*(0)$. If it is assumed true for h then it follows immediately for $h + 1$ using the inductive definition of $C \cap \mathsf{Damage}((q+1)T)$ given in Section 13.4, the definition of $\mathcal{N}^*$ and Lemma 14.7. □

To make the triple $(x^*, \mathcal{N}^*, \mathsf{Damage}^*(0))$ a self-correcting evolution we have to make it satisfy the Restoration Condition and the Computation Condition.

The Restoration Condition says now that at all times hT^*, all big colonies disjoint from $\mathsf{Damage}^*(hT)$ are legal. We will prove the following lemma.

LEMMA 16.2. *There is a universal medium* Univ *supporting colonies with the following property. Assume for parameters* $P', \ldots, r$, *the Size Condition 13.3. Let* h *be a natural number. Assume that at time* hT^*, *for all big colonies* E *not intersecting with* Damage*(hT), *the following conditions are satisfied.*

(*a*) *In each row of the alliance* E_*, *the deviations of the InpMem track of evolution* x *from the evolution* y [*as defined in* (16.1)] *are covered by* $\langle \text{dev} \rangle r$ *T-intervals.*

(*b*) E *is legal.*

Then the same is true at time $(h+1)T^*$ *for all big colonies not intersecting with* Damage*$((h+1)T^*)$.

For $h = 0$, the assertions $(a - b)$ in the lemma will be satisfied, due to the definition of Damage*(0) in 16.1 above. Therefore this lemma will imply that these conditions are satisfied for all h, i.e., the Restoration Condition holds.

Let C be a big cluster in W^* containing the big colony E. If the noise $\mathcal{N}^*$ is not empty during the interval $[T^*; h]$ in $C_{hU} = \Gamma(C, T^*)$ then C belongs to Damage*$((h+1)T^*)$, and the conclusion of Lemma 16.2 is automatically satisfied. Suppose now that the noise $\mathcal{N}^*$ is empty in this region and $\mathcal{I}$ is a minimal quarantine for C at time hT^*. Then by the definition of Damage(qT) in Section 13.4, we have

$$C \cap \text{Damage}^*((h+1)T^*) = C \cap \bigcup D(\mathcal{I}, P^* + T^*).$$

We must design Univ therefore in such a way that if E does not intersect $D(\mathcal{I}, P^* + T^*)$ then properties $(a - b)$ are satisfied in E at time $(h+1)T^*$.

16.3. Initialization

Our original definition of the medium Univ will not guarantee that the big colony E becomes legal if it was not. Indeed, even in the absence of noise, the small colonies just simulate the evolution of whatever they find in the big colony and its neighbor. The simulated medium is now Univ again. Though we made Univ to support colonies, this property is usable only for a big colony that is already healthy.

To make the simulated big colony healthy "against its own will," a simple but crucial new procedure, called *Init*, will be inserted between the procedures *Input* and *Compute*, at the beginning of each big work period. After the procedure *Input*, the configurations of these colonies are in nine subsquares of size $QP/3$ of the alliance, on the Simulator track. Let us agree that whenever we speak about a big colony we mean one of these nine subsquares of the Simulator track of our alliance.

The procedure *Init* puts the symbol *topleft* into the left-most column and the top row of the simulated big colony and checks that all other cells of this colony are in a state belonging to S_{init}. If a cell is in some other state then its state is changed arbitrarily to one in S_{int}. Finally, *Init* writes the integers Q', U' in place of the first two parameters in the parameter field. (They play the role of P', T' for the big colony).

Now the new procedure *CompColumn* has the following form.

procedure *CompColumn*(s, t, l);
begin
  Decode;
  Input;
  if modif $=$ 1 **and** $l = 1$ **then** *Init*;
  Compute(t);
  Encode;
  Output(s);
end;

It is crucial that in the procedures *ForceCode* and *Init*, the small colonies intervene in the simulation of big colonies. The code φ thus defined is therefore no more a pure simulation, since health is *forced* on the big colonies. Since, however, φ is not a simulation, we do not want to use this program always, and the parameter modif allows us to make the choice.

Sections 15 and 16.3 engineered the "magic" recovery needed for the big colonies to recover from loss of structure. The rest of the present section proves that the magic works.

16.4. Legalization Works

Let E be a big colony disjoint from $D(\mathcal{I}, P^* + T^*)$. We have to show at time $(h + 1)T^*$, conditions $(a - b)$ of Lemma 16.2 hold. The following geometrical lemma is easy to verify.

LEMMA 16.3. *Let $\mathscr{H}$ be a set of disjoint triangles. Let E be a P^*-square disjoint from $D(\mathscr{H}, P^*)$. Let the P^*-squares E_1, E_2, E_3 be the northern, southeastern, and southwestern neighbors of E. At least two of these three squares are disjoint from $\mathscr{H}$.*

We apply this lemma to $\mathscr{H} = \mathscr{I}^1$ as defined above and the big colony E that we assumed at the beginning of the previous paragraph to be disjoint from $D(\mathscr{I}^1, P^*)$. It follows that two of the three neighbors E_i are disjoint from $\mathscr{I}_1$. Without loss of generality, we can assume that E_1 and E_2 are disjoint from $\mathscr{I}^1$.

We assumed that $\mathscr{I}$ is a quarantine at time hT^* for the evolution x^*. Hence by Lemma 16.1, $\mathscr{I}$ is a local quarantine at that time, and E_1, E_2 are legal big colonies at the same time.

We will now follow the reasoning of Section 14.4. Before time $(h+1)U$, only a singular event can create new deviations on the *InpMem* track. According to Lemma 14.8, if cluster F is the site of a singular event then its neighborhood $\Gamma(F, T)$ intersects

$$D((\mathscr{I}^1 \cup \mathscr{K} \cup \mathscr{L})', T).$$

The alliances E_{1*}, E_{2*} are disjoint from $\mathscr{I}^1$ at time hUT. Just as in the proof of Lemma 14.6, we can conclude the following lemma.

LEMMA 16.4. *During the time interval $[hUT..(h+1)UT)$, in each row of E_{1*} and E_{2*}, the number of T-intervals with deviations on the InpMem track is bounded by $\langle \text{corr} \rangle r$.*

The following lemma says that after time $(hU + 2Q)T$, in the alliances E_*, E_{1*}, and E_{2*}, the singular events are restricted as in the proof of Lemma 14.6.

LEMMA 16.5. *If cluster F in $E_* \cup E_{1*} \cup E_{2*}$ is singular at time qT for q in $[(hU + 2Q)T..(h+1)U)$ then $\Gamma(F, T)$ intersects $\mathscr{L}^0$ and q is in Y.*

Proof. Remember $P^* = PQ$. According to Lemma 14.8, the neighborhood $\Gamma(F, T)$ is intersected by $\mathscr{H}_q^0 \cup \mathscr{L}^0$. Here, $\mathscr{H}_q = D(\mathscr{I}^1, (q - \langle \text{kill} \rangle)P)$. Let us show that for $q > hU + 2Q$ the set $\mathscr{H}_q^0$ is disjoint from $\Gamma(F, T)$. This will imply that the latter intersects $\mathscr{L}^0$. Lemma 14.8 says that in this case, q is in Y.

We have

$$(q - \langle \text{kill} \rangle)P > (2Q - \langle \text{kill} \rangle)P$$
$$= PQ + (Q - \langle \text{kill} \rangle)P \geqslant PQ + 2T.$$

Here we used $(Q - \langle \text{kill} \rangle)P > 2T$, which follows from the Size Conditions. It follows that $H_q^0 \subset D(\mathscr{I}^1, PQ + 3T)$. We assumed that $D(\mathscr{I}^1, PQ)$ is disjoint from E_*, E_{1*}, and E_{2*}, hence it is disjoint from F. Therefore by (11.2), the neighborhood $\Gamma(F, T)$ is disjoint from $D(\mathscr{I}^1, PQ + 3T)$. □

Proof of Lemma 16.2. The program begins with $2Q$ idling steps. After these, according to Lemma 16.5, the singular events are restricted in E_*, E_{1*}, and E_{2*} just as stated by Lemma 14.2.

We assumed modif $= 1$. Hence for $l = 1$, in all regular calls s of the procedure *CompColumn*(s, t, l), the procedure *Init* makes the simulated colony healthy. The simulated medium Univ, by assumption, supports colonies. Therefore since the big colonies E_1, E_2 were legal, by the end of the procedure *Compute*(t), the result represents a legal colony.

The code φ is such that the top three rows of the alliance code the top three rows of the corresponding big colony. Therefore the top three rows of the alliance deviate in at most $\langle \text{dev} \rangle r$ T-intervals from the code of the top three-rows of a legal big colony.

Finally, the procedure *ForceCode* decreases CodeDiff(u_k) to $\langle \text{dev} \rangle r$, for all $k > 2$. □

End of the proof of Theorem 16.1. To make the evolution $(x^*, \mathscr{N}^*, \text{Damage}^*(0))$ self-correcting, it remains to prove the Computation Condition. This condition says now the following: Let the big cluster C be regular at time hT^*. Then the configuration $x^*[(h + 1)T^*, C]$ is what it would be if x^* was a trajectory of Univ starting from the same configuration $x^*[hT^*, C_{hU}]$.

The regularity of the big cluster C at time hT^* means a condition on the noise, and one for the damage. For the noise, it says that the noise $\mathscr{N}$ is sparse over the set $[UT; h] \times C_{hU}$ in W. For the damage, it says that Damage$^*(hT^*)$ does not intersect C_{hU}. By the definition of Damage$^*(0)$ and Lemma 16.2, this implies that each row of each alliance of C_{hU} differs from a codeword by at most $\langle \text{dev} \rangle r$ T-intervals.

Now Lemma 14.2 has essentially these conditions and the needed conclusions. □

17. CONSTANT SPACE REDUNDANCY

In the present section, we sharpen Theorem 13.1, by weakening one of the Size Conditions 13.3. This condition has the form

$$QP' > \max(3Q'|\mathsf{Med}_0|,\ Q'|\mathsf{Med}_0| + 9\langle\text{corr}\rangle rP'T/P).$$

We want to eliminate the factor $3|\mathsf{Med}_0|$. This economy is necessary only when our goal is constant space redundancy, as promised in the main theorem. In that case, the factor $3|\mathsf{Med}_0|$ cannot be tolerated, since the final simulation is a concatenation of several simulations of this kind, multiplying the cell size (which is the measure of the space redundancy) every time by $3|\mathsf{Med}_0|$.

As long as we design the simulation it is convenient to use unit cell sizes and cell worktimes in the simulated colonies $[Q'; i, j]^2$. The new cell size α^* and cell worktime β^* can then be computed at the end, as done in Section 16. These parameters do not enter the Size Conditions anyway.

17.1. Economical Trajectories

If we want to get rid of the factor $|\mathsf{Med}_0|$ then most cells of the Q'-square should not carry much information so that an encoding into binary strings of length $|\mathsf{Med}_0|$ is unnecessary. Therefore the largest part of the Q'-square will be destined to the passive storage of information, i.e., it will be "memory." We add more ingredients to our model. We assume that there is a four-element subset $S_{\mathsf{Med}_0,\text{mem}}$ of the states of the medium Med_0, called the set of *memory states* that is *invariant* with respect to the rule Med_0, i.e., the rule Med_0 never changes a memory state into a nonmemory state.

We also assume that a parameter $\langle\text{wksp}\rangle' \geqslant 1$ is given. A Q'-square of Med_0 will be called *economical* if its cells in all but the bottom $Q'/\langle\text{wksp}\rangle'$ rows have memory states. These bottom rows will be called the Workspace field of the Q'-square, the rest the Memory field. Notice that the notion of an economical square depends on the parameter $\langle\text{wksp}\rangle'$ and the set $S_{\mathsf{Med}_0,\text{mem}}$.

In Theorem 13.1, we simulated all possible trajectories of Med_0. Now we confine ourselves to trajectories that are *economical*, i.e., such in which all colonies are economical. Due to the invariance of the set of memory states, if a trajectory starts from a configuration of economical colonies, then it is economical.

Though we make our task easier by confining ourselves to the simulation of economical trajectories, we pay for this by making the medium Univ and its simulating evolutions also economical. The set S_{mem} of memory states of Univ will be part of the set S_{init} of initial states introduced in Section 13.3. There will also be a parameter $\langle \text{wksp} \rangle$ for the definition of economical small colonies. The small colonies will be required to be both legal and economical.

The set of parameters of our model is now

$$P', T', Q, U, Q', U', P, Q, \langle \text{wksp} \rangle, \langle \text{wksp} \rangle', \mathsf{Prog}_0, \mathsf{modif}, r.$$

We replace the Size Conditions 15.1 with the following.

CONDITION 17.1 (SIZE).

$$QP'\left(1 - \frac{1}{\langle \text{wksp} \rangle}\right) > Q'\left(1 + \frac{|\mathsf{Med}_0|}{\langle \text{wksp} \rangle'}\right) + 9\langle \text{corr} \rangle' r P' T / P,$$

$$P' > \max(\log U, \log T', |\mathsf{Prog}_0|),$$

$$UT' > (\langle \text{wksp} \rangle |\mathsf{Med}_0|)^2 \times \frac{U'}{Q'}\left(\frac{QP}{T}\right)^2 QT' \log(QP')\mathsf{Step}_0,$$

$$T' \geqslant 9P',$$

$$P \mid T, \qquad \frac{T}{P} \,\Big|\, Q, \qquad Q \mid U.$$

Only the bounds on QP' and UT' have changed. The bound on QP' is smaller: it will be asymptotically equal to Q'. On the other hand, we pay with a factor $(\langle \text{wksp} \rangle |\mathsf{Med}_0|)^2$ in the bound on the time UT', for the fact that only a fraction $1/\langle \text{wksp} \rangle$ of each colony is devoted to workspace.

Now we can formulate the optimized version of Theorem 13.1.

THEOREM 17.1. *There is an economical medium* Univ *supporting colonies such that the following holds. Assume the Size Condition 17.1 for some economical* Prog_0, *and numbers* $P', \ldots, r$. *Then there is a standard simulation* φ *of* Med_0 *by* Univ *mapping economical trajectories into economical trajectories, such that the conclusion of Theorem 13.1 holds.*

The rest of the present section is devoted to the proof of the above theorem. It describes all necessary modifications to the medium Univ and the code φ. If the reader is not interested in this optimization then this section can be skipped.

17.2. Drying

Ordinary self-simulation would proceed by expanding the content of each cell to be simulated, as done at the beginning of Section 13 and storing the result in the Memory of the simulating colony. It is possible to avoid expansion by giving different treatment to the Memory and the Workspace. Due to the small number of memory states, only the Workspace must be expanded. The operation can be viewed as *drying* a flower by pressing it between the leaves of a book. The process will not change the parts that are already dry.

Formally, one can define a code

$$\mathsf{Dry} = (\mathsf{Dry}_*, \mathsf{Dry}^*).$$

It is applied to a Q'-square and encodes it into an array of memory cells. The code *leaves the Memory unchanged* and expands each symbol of the Workspace into a column $\mathsf{bin}(s)$ of height $|\mathsf{Med}_0|$. Thus, each row of the Workspace is expanded into $|\mathsf{Med}_0|$ rows. The vertical size of the Q'-square increases therefore to

$$Q'(1 - 1/\langle \mathrm{wksp} \rangle') + |\mathsf{Med}_0| Q'/\langle \mathrm{wksp} \rangle'.$$

The crucial difference between drying and the error-correcting code Algeb is that since drying is symbol-for-symbol, the simulating alliance can manipulate the dry information *without ever reconstituting it*. The parameters of the alliance contain $\langle \mathrm{wksp} \rangle'$, hence the small colonies of the alliance will know which simulated cells are expanded and which ones are not.

17.3. The Code φ

The Memory field of a Univ-colony is divided into a vertical strip of width $P/\langle \mathrm{wksp} \rangle$ on the right side, called the Channel field, and the rest, called the Data field. The Channel field is needed only for communication between the Workspaces of the colony and its northern neighbor. Only the Data field, which still occupies most of the colony, will be used for actual storage.

The union of the Memory fields of the member small colonies will be called the *Memory field of the alliance*. The Data, Workspace, and Channel fields are defined similarly. Thus, the Workspace and Channel fields of the alliance form a grid: a union of horizontal and vertical strips of width $P/\langle \text{wksp} \rangle$. The Data field is the complement: it consists of Q^2 squares of size $P(1 - 1/\langle \text{wksp} \rangle)$.

Let us define the code φ. The encoding φ_* is defined as follows. First we apply the code $\mathsf{Algeb}_* \circ \mathsf{Dry}_*$ to the Q'-square. Then we distribute the result in the Data fields of the small colonies of the alliance. Finally, for all i, j, we make each of the small colonies legal, initialized to the beginning of a work period of the alliance.

The decoding φ^* is defined as follows. For each row of the alliance intersecting with the Data, we take the information from the Data parts of the row. We apply the decoding $\mathsf{Dry}^* \circ \mathsf{Algeb}^*$ to the rectangle obtained from all rows this way. This means reconstituting the Workspace rows of the Q'-square from their expanded form.

17.4. Memory in the Universal Medium

How will we retrieve information from the Memory field of Univ-colonies? For $n = 0, 1, 2, 3$, if a cell has the memory state s_n then we say that this cell stores the pair of bits n_0, n_1 in n. Memory cells with shift their left bits up and their right bits down between each other. Thus, if x is a trajectory of Univ and

$$x[t, i, j] = s_{m_1 m_0}, \qquad x[t, i+1, j] = s_{n_1 n_0}$$

for bits m_0, m_1, n_0, n_1 then we will have

$$x[t+1, i, j] = s_{n_1 u}, \qquad x[t+1, i+1, j] = s_{v m_1}$$

for some bits u, v. If the upper neighbor is not a memory cell then the cell reads some fixed function of the state of the upper neighbor into its right bit: the effect of this is that a workspace cell can "write" into a memory cell. (Of course, it can also read.)

During almost the whole computation, the workspace cells at the upper and lower edges of the Memory have states that keep the information rotating in each column: flowing down in the right bits and turned back at the lower edge to flow back up in the left bits. An appropriate times, the workspace cells can write something into the Memory or read from it.

17.5. Ranks and Slots

We must give up the luxury of separate *InpMem* and *OutMem* tracks in Univ. Moreover, since the Workspace is only a small part of the whole colony, we cannot store all nine coded neighbor big colonies in the alliance for simulation. We solve these problems in the present subsection by *trading time for space.*

Let

$$R = \left\lfloor \frac{QP'}{7|\mathsf{Med}_0|\langle \text{wksp} \rangle} \right\rfloor.$$

Let us divide the Q'-square into horizontal strips called *ranks.* The width of each rank is at least R but smaller than $2R$. Each rank is either completely in the Memory or completely in the Workspace. These are the only requirements for ranks, and it is easy to find a subdivision with these properties. With the Memory, e.g., we can begin to divide it into strips of width R, leaving at the bottom a somewhat wider last strip.

The computation will update the content of the simulated Q'-square rank-by-rank. An *extended rank* is a rank extended by its left and right neighbor ranks from the corresponding neighbor colonies. After drying, a rank of width n will be represented by n or $|\mathsf{Med}_0|n$ rows depending on whether it comes from the Memory or the Workspace. The number R is determined in such a way that even when they have $2R|\mathsf{Med}_0|$ rows after drying, three neighboring ranks will fit comfortably into the Workspace of the alliance. To fit there three neighboring extended ranks, we just store three symbols per cell of the simulating medium Univ. (Each symbol consists of at most two bits.)

Let

$$N = \lfloor U'/R \rfloor.$$

The computation will proceed in *stages* $l = 1, \ldots, N + 1$. Each but the last stage computes the state of the Q'-square after R more time units. Some care must be taken not to update information that will be needed in the next stage of the serial simulation. One solution, taken from Toffoli's Cellular Automaton Machine (see [Tof]), is to distinguish between the physical and logical positions of a rank. The physical position of a rank will be called its *slot.* It will vary

from stage to stage. Each slot is a strip consisting of a sufficient number of rows in the simulating alliance, to store a certain rank. The number

$$m = \lfloor QP'(1 - 1/\langle \text{wksp} \rangle) \rfloor$$

of rows in the Memory of the simulating alliance is somewhat more than needed to accommodate all ranks. Indeed, the number of all rows in the dried Q'-square is less than $Q'(1 + |\mathsf{Med}_0|/\langle \text{wksp} \rangle')$, which, according to the Size Conditions, is still less than m.

We will use the m available rows for slots as a circular store. At the beginning, $slot(i, 0)$ holds the initial content of rank i. These slots are consecutive strips starting from the top row of the alliance. Each stage l of the simulation consists of *substages* i, corresponding to the ranks of the alliance. In stage l, at the beginning of substage i, the new content of rank j for $j \leqslant i - 1$ is found in $slot(j, l)$. The old content of ranks $j \geqslant i - 1$ is found in the slots $slot(j, l - 1)$. In the cycle of available rows $\bmod m$, the slots $slot(j, l - 1)$ for $j \geqslant i - 1$ are followed by $slot(j, l)$ for $j \leqslant i - 1$. The union of all these slots forms a segment $\bmod m$.

In substage i, we load the old content of the extended ranks $i - 1$, i, and $i + 1$ into the Workspace of the alliance from the corresponding slots. The simulation is performed (with the repetitions s as earlier), and the new value of rank i is stored in a new slot $slot(i, l)$ at the end of the segment. At the same time, $slot(i - 1, l - 1)$ is released, since it is no longer needed.

17.6. Procedures

The above outline results in the following procedures. Let $\langle \text{ranks} \rangle$ denote the number of ranks.

procedure $Input(i, l)$

for $i = 2, \ldots, \langle \text{ranks} \rangle - 1$ and $l = 1, \ldots, N + 1$ loads the Data of the extended slots

$$slot(i - 1, l - 1),\ slot(i, l - 1),\ slot(i + 1, l - 1)$$

(representing the corresponding extended ranks) and spreads it in the Workspace of the alliance. For $i = 1, \langle \text{ranks} \rangle$, the information

of one of the three ranks must be taken from the appropriate slot of the upper resp. lower neighbor alliance. For extended rank i, for a row n in one of the above three slots, its destination is a row

$$Map(i, n)$$

in the destination alliance. The function $Map(i, n)$, as well as the boundaries of all slots in all stages, could be described by explicit formulas. The Workspace occupies only a portion $1/\langle \text{wksp} \rangle$ of each small colony. Therefore at its destination given by the function $Map(i, n)$, the information of a rank will occupy $\langle \text{wksp} \rangle$ times more colony rows than in its slot.

procedure *Output* (i, s, l)

takes the information in rows $Map(i, n)$ of the colonies in column s of clusters and writes it back to the rows n of column s of clusters in $slot(i, l)$.

After stage $l = N + 1$,

procedure *Backrotate*

rotates the alliance vertically back to the initial state when rank i is stored in $slot(i, 0)$. This procedure is just a series of copying operations. It cannot carry deviations from one column to another.

procedure *Decode*

decodes the Data loaded into all rows $Map(i, n)$ using the code Algeb described in Section 8. Procedure *Encode* performs the corresponding encoding. After encoding as well as decoding, the information is left in the same row.

procedure *Init* (i)

depends now on i, since the top row of the simulated Q'-square is located in a slot dependent on i.

procedure *Compute* (t, i)

simulates t steps (less than the width of the rank after decoding) of the work of the three decoded extended ranks. Even after decoding,

the information is still in dry form. Since the code Dry does not treat all ranks the same way (the higher numbered ranks belong to the Workspace of the Q'-square), procedure *Compute* depends on i.

procedure *ForceCode*

remains much as it was defined in Section 15, only it will also have to work rank-by-rank, due to space shortage.

17.7. The Program

Small colony support can be added to the program below just as easily as in Section 16.3.

```
begin
  for l = 1 to N + 1 do begin
    if l ⩽ N then t := R else t := U' − NR;
    for i = 1 to ⟨ranks⟩ do begin
      for s = 1 to QP/T do begin
        Input(i, l);
        Decode;
        if modif = 1 and l = 1 then Init(i);
        Compute(t, i);
        Encode;
        Output(i, s, l);
      end;
      ForceCode;
    end;
  end;
  Backrotate;
end.
```

It is easy to see that the procedures of this program can be written now in such a way as to satisfy the new Size Conditions. Essentially, the only new Size Condition to check is the bound on UT'. The main difference of the present program from the earlier one is that what was earlier an U'/Q'-fold iteration, is now replaced with an iteration U'/R-fold, combined with an iteration ⟨ranks⟩-fold. The extra factor is therefore

$$\langle \text{ranks} \rangle Q'/R \leqslant (Q'/R)^2 \leqslant (|\text{Med}_0| \langle \text{wksp} \rangle)^2.$$

The proof of Theorem 13.1 can now be almost literally carried over to the proof of Theorem 17.1 when the program there is replaced with the present one. □

If we restrict ourselves to economical trajectories of Univ then Theorem 16.1 will also remain true with the new Size Conditions, and the proof carries over virtually without change.

18. PROOF OF THE MAIN THEOREM

18.1. A Series of Implementations

For $k = 0, 1, \ldots$, let us generate a sequence of parameter sets $P', \ldots, r$ each of which satisfies the Size Conditions 17.1. For the reader who skipped Section 17: a sequence satisfying the earlier Size Conditions 15.1 is even easier to construct. The only difference is that P'_k/P'_{k-1} will be about 3|Univ| instead of converging to 1.

The noise bound r_k and the probability p_k are defined as

$$r_k = 2^k, \qquad p_k = \varrho^{r_1 \cdots r_k}.$$

Let w be an integer parameter to be chosen conveniently large later. Let us define the basic colony size P_0 and, for $k \geqslant 0$, the auxiliary parameter g_k as follows.

$$\begin{aligned} P_0 &= g_0 = w, \\ g_k &= (wk)^{11} r_k^2 g_{k-1} \qquad \text{for } k > 0 \end{aligned} \tag{18.1}$$

The absolute and relative colony sizes and work periods P_k, T_k, Q_k, U_k are defined as follows.

$$\begin{aligned} T_k &= P_k g_k \\ P_k &= (wk)^2 r_k T_{k-1}, && \text{for } k > 0, \\ Q_k &= P_k/P_{k-1} = (wk)^2 r_k g_{k-1} && \text{for } k > 0 \\ U_k &= T_k/T_{k-1} = (wk)^2 r_k g_k && \text{for } k > 0. \end{aligned}$$

Finally, the represented colony sizes and work periods are defined as follows.

$$P'_0 = w, \qquad P'_1 = \lfloor Q_1/2 \rfloor,$$

$$P'_k/P'_{k-1} = (1 - k^{-2})Q_k \qquad \text{for } k > 1,$$

$$T'_k = wP'_k.$$

THEOREM 18.1. *For large enough w and small enough ρ, there is a series of implementations Ψ_k for $k > 0$, with parameters P'_k, $T'_k, P_k, T_k, \varrho, p_k$.*

Proof. Let us fix k. We define the parameters

$$P' = P'_{k-1}, T' = T'_{k-1}, P = P_{k-1}, T = T_{k-1},$$

$$Q = Q_k, U = U_k, Q' = P'_k, U' = T'_k,$$

$$\langle \text{wksp} \rangle = wk^2, \langle \text{wksp} \rangle' = w(k+1)^2, r = r_k.$$

We prove that if w is chosen large enough and ϱ small enough then the above choices satisfy the Size Conditions 17.1 with $\mathsf{Med}_0 = \mathsf{Univ}$, and for $k > 1$ the inequality

$$p_{k-1} \leqslant U_k^{-3(r_k+1)}. \tag{18.2}$$

[This inequality (18.2) corresponds to (9.1).] Then, the application of Lemma 13.1, and Theorem 16.2 completes the proof.

Proof. The first Size Condition can be written as

$$QP(1 - 1/\langle \text{wksp} \rangle) > Q'(1 + |\mathsf{Univ}|/\langle \text{wksp} \rangle')\alpha + 9\langle \text{corr} \rangle' rT. \tag{18.3}$$

We have $Q'\alpha = P'_k P_{k-1}/P'_{k-1} = (1 - k^{-2})P_k$. Therefore the right-hand side of (18.3) can be written as

$$P_k(1 - k^{-2})\left(1 + \frac{|\mathsf{Univ}|}{w(k+1)^2}\right) + 9\langle \text{corr} \rangle' P_k (sk)^{-2}$$

$$< P_k(1 - k^{-2}(1 - |\mathsf{Univ}|/w - 9\langle \text{corr} \rangle'/w^2)).$$

Suppose that w is large enough to have

$$|\mathsf{Univ}|/w + 9\langle \mathrm{corr}\rangle'/w^2 < 0.5.$$

Then the right-hand side of (18.3) is less than

$$QP(1 - 1/\langle \mathrm{wksp}\rangle) = P_k\left(1 - \frac{1}{wk^2}\right).$$

The second Size Condition says, with the substitutions made:

$$P'_{k-1} > \max(\log U_k, \log T'_{k-1}, |\mathsf{Prog}_0|).$$

Here, Prog_0 is the program of the transition function Univ on the Turing machine Turing. The condition $|\mathsf{Prog}_0| < P'_{k-1}$ follows if we set

$$w > |\mathsf{Prog}_0|.$$

We have $\log T'_{k-1} = \log P'_{k-1} + \log w < P'_{k-1}$ since $P'_{k-1} \geqslant w$.

Before going further, let us find some explicit formulas and estimates from the above recursive definitions. We have, with functions $0 < \lambda(k, i) < 1$,

$$g_k = w^{11k}(k!)^{11}2^{2\binom{k+1}{2}} \tag{18.4}$$

$$= \exp_2\left\{2\binom{k+1}{2} + 11k(\log(wk) - 2\lambda(k,1))\right\}, \tag{18.5}$$

$$Q_k = (wk)^{-9}g_k,$$

$$P_k = w^{-9k}(k!)^{-9}g_1 \cdots g_k$$

$$= \exp_2\left\{2\binom{k+2}{3} + 11\binom{k+1}{2}\log(wk)\lambda(k,2)\right\}.$$

The relations

$$\log U_k < 0.5Q_{k-1},$$

$$\log P_k < wk^3 \tag{18.6}$$

are now easy to check. The first one implies

$$\log U_k < 0.5 Q_{k-1} < P'_{k-1},$$

which finishes the proof of the second Size Condition. The third Size Condition reads:

$$U > (\langle \text{wksp} \rangle |\text{Univ}|)^2 (U'/Q')(QP/T)^2 Q \log (QP') \text{Step}_0. \tag{18.7}$$

We have

$$\begin{aligned} U'/Q' &= T'_k/P'_k = w, \\ (QP/T)^2 &= (P_k/T_{k-1})^2 = (wk)^4 r_k^2, \\ QP' &= Q_k P'_{k-1} < P_k = Q_k P_{k-1}, \\ UT' &= U_k T'_{k-1} = w U_k P'_{k-1} > U_k P_{k-1}. \end{aligned}$$

Hence the right-hand side of (18.7) can be estimated, using (18.6), as

$$\begin{aligned} &(wk^2 |\text{Univ}|)^2 w (wk)^4 r_k^2 Q_k \log P_k \text{Step}_0 \\ &< |\text{Univ}|^2 w^7 k^8 r_k^2 Q_k w k^3 \text{Step}_0 \\ &< (wk)^{11} r_k^2 Q_k = U_k. \end{aligned}$$

This proves the third Size Condition. The remaining two Size Conditions are satisfied by definition.

To prove (18.2), we prove the following stronger inequality, with $R = -\log \varrho$.

$$\log U_k \leqslant r_1 \cdots r_{k-1} R/4 r_k = R 2^{\binom{k-1}{2}-3}.$$

We have, from (18.4):

$$\log U_k = \log g_k + k + 2 \log (wk) < 2 \binom{k+1}{2} + 12k \log (wk).$$

This is clearly smaller than the right-hand side of the previous equation, if only R is not too small relative to w. □

18.2. Divisibility Considerations

Let us return to the formulation of the Main Theorem. Given the values n, t, ε, let K be the smallest k with

$$(n + P_k)^2 t\varrho^{r_1 \cdots r_k} < \varepsilon.$$

It is convenient to assume that n is divisible by P'_K. Let us show that if this is not the case then a simple extra coding η of Med_0 into a slightly different medium Med_1 will achieve this. No subtlety is involved, and we mention the details only for completeness' sake. We have seen in Section 15 that a medium on a torus can be treated as it were folded to a medium working on a square, using the mapping $(i, j) \to (\tilde{\imath}, \tilde{\jmath})$. If we surround the square by "blanks" then the boundaries of the square are automatically marked. And there is no difficulty in simulating a medium on a square by a medium on a larger square. Let

$$n' = P'_K \lceil n/P'_K \rceil \tag{18.8}$$

be the smallest multiple of P'_K greater than n. Thus, using a step-for-step simulation η we order to each configuration of Med_0 over $\mathbf{Z}_n^2$ a configuration of a new medium Med_1 over $\mathbf{Z}_{n'}^2$.

18.3. Deviation Probability

The simulation γ of the theorem will be given as

$$\gamma = \psi_{K-1} \circ \varphi \circ \eta.$$

Here, η is the simulation described in Section 18.2. The simulation ψ_{K-1} is provided by Theorem 18.1. The simulation φ with parameters P'_K, P_K, T'_K, T_K of the medium Med_1 by Univ is given by Theorem 13.1, with the parameters as at the beginning of the proof of Theorem 18.1, with $k = K$. The target space of code γ is $\mathbf{Z}_m^2$ with

$$m = \frac{n'}{P'_K} \frac{P_K}{T_0}. \tag{18.9}$$

We divide by T_0 since in the implementation $\mathbf{\Psi}_0$, the size of a single cell of medium M was chosen T_0.

The number t of steps of the original computation will be followed for $\lceil t/T'_K \rceil$ work periods of the simulation.

Now let z be a trajectory of Med_1, then $y = \varphi_*(z)$ is a trajectory of Univ. Let ξ be a ϱ-perturbation of $\psi_{(K-1)*}(y)$. Then by the definition of implementations, the random evolution $\psi^*_{K-1}(\xi)$ is a self-correcting p_{K-1}-perturbation of y. In each T_K-cube, the probability that the noise belonging to this p_{K-1}-perturbation is not (U_K, r_k, T_{k-1})-sparse, is at most $p'^k_{K-1} = p_k$. The number of T_K-cubes in the space–time area in question is smaller than $n'^2 t$. It follows that the probability that over this area the noise is not sparse is at most $n'^2 t p_K < \varepsilon$. With probability $1 - \varepsilon$, the noise is sparse. Using Theorem 13.1 we conclude that with probability $1 - \varepsilon$, we have $\varphi^*(\psi^*_{K-1}(\xi)) = z$.

18.4. Redundancy

First we prove

$$P_K, T_K = O(2^{(\log L)^{1.6}}). \tag{18.10}$$

From (18.1), it follows that the quantities g_k, U_k grow approximately as 2^{k^2}:

$$g_k = 2^{k^2 + O(k \log k)},$$

$$U_k = 2^{k^2 + O(k \log k)}.$$

Therefore the quantities P_k, T_k grow approximately as $2^{2k^3/3}$:

$$\begin{aligned} T_k &= 2^{2k^3/3 + O(k^2 \log k)}, \\ P_k &= 2^{2k^3/3 + O(k^2 \log k)}. \end{aligned} \tag{18.11}$$

Above, K was defined as the smallest k with

$$2 \log_\varrho [(n + P_k)^2 t/\varepsilon] \leqslant r_1 \cdots r_k.$$

If $n > P_K$ we get from here, with $L = \log(n^2 t/\varepsilon)$ as defined in the Theorem:

$$L = 2^{K^2/2 + O(K \log K) + O(1)}.$$

The constant in $O(1)$ here depends on ϱ. If $n \leqslant P_K$ the same estimate is obtained. From this, and (18.11), we obtain (18.10).

Now we prove $m = O(n + P_K)$. It follows from (18.8) that $n' = O(n + P_K')$. Now it follows from (18.9) that $m = O(n + P_K)(P_K/P_K')$. Finally, it follows from the definitions (18.1) that

$$\frac{P_k}{P_k'} = O\left(\prod_{k>1} \frac{1}{1 - 1/k^2}\right) = O(1).$$

The definitions (18.1) show that the time redundancy is proportional to g_k. By the above equation, this is

$$2^{O(K \log K)} L^2 = L^{2+o(1)}.$$

The third statement of the theorem concerns the complexity of computation of the code. It was proved in Section 8. □

19. CONCLUSION AND OPEN PROBLEMS

19.1. Simplification

Simplification can be understood in conceptual and in quantitiative sense. In order to lend the two-dimensional medium physical credibility, reducing the number of states (while retaining nearest-neighbor interaction) is crucial. The features required at the cell level include

- computational universality;
- parallel transport with periodic copying;
- application of a generalized Toom's Rule to some local variables (the phase variables);
- periodic majority vote among certain local variables.

These features can probably be achieved by a relatively simple local rule. The elaboration of details would be very interesting.

19.2. Continuous Time

The hierarchical organization used for error correction can also be used for synchronization. Each block of cells is supposed

to be synchronized to a close tolerance and is periodically resynchronized. Higher order blocks are synchronized to progressively looser tolerances. Details must be worked out yet.

If worked out for the one-dimensional case, the result will be a formal refutation of the Positive Rates Conjecture featured in [Li].

For the three-dimensional case, Charles H. Bennett has a physical synchronization idea. If something of comparable simplicity does not work, then the simplicity of the construction of [GR] will be lost with the introduction of continuous time.

19.3. Space–Time Trade-off

This problem was discussed in Section 17. We do not consider the logarithmic lower bound of [DO1] significant as a lower bound on the redundancy of reliable computation. It shows only the necessity of encoding. But in our view, for the purposes of reliable computation, information must be viewed as *coming in encoded form*.

It is remarkable (though probably an artifact) that both here and in [GR], the product of the space and time redundancies came out $\log^2 N$. The bottlenecks in the present paper that caused $\log^2 N$ instead of $\log N$ occur in the problem of consensus. In Section 11.3 more than r^2 repetitions were required for r possible failures. We hope that this result can be improved to $O(r)$ repetitions. However, even more computation was needed in the procedure *FindVotes* that found reliably the votes whose consensus is sought.

19.4. Self-Organization

Our reliable media work reliably only if the starting configuration already contains the (input-independent) hierarchical organization. It would sound more natural, in one dimension, that if the input is one bit then the starting configuration should consist just of the repetition of this one bit. If error correction requires structure then the medium should be able to build up this structure, out of "nothing."

In two dimensions, remembering one bit starting from a homogeneous starting configuration can be done by Toom's Rule. But there is still the problem of *increasing depth in the presence of noise*, as outlined in Section 2, starting from a homogeneous configuration. This is what we consider the natural problem of self-organization.

This goal is the *most intriguing* among the ones proposed. Let us note first of all that Toom's Rule must be largely abandoned even

in two dimensions since its usefulness for structure maintenance depends on a globally existing structure. We must therefore return to the ideas of [G]. Still, we run into new problems at several points.

- Without errors, no structure can arise, since all cells have the same state and the same rule. Hence, randomness is no more than just an "enemy": whatever structures may arise will be random (e.g., if there are blocks, their position will be random).
- Killing a small organized island surrounded by inconsistencies was one of the main principles of the construction in [G]. Self-organization requires that we permit these islands to grow, at least very slowly. A possible new rule replacing the old one could be that islands die if they are prevented from growth for longer time.
- More probabilistic analysis is required with any of these ideas, to show that there will always be islands not too far from the origin whose size is greater than some increasing function of time.

Self-organization is, of course, a favorite topic of theoretical biology. What distinguishes our approach is that all the structure in [G], however life-like, seems to be forced upon us by the extremely simple requirement: the protection of information in a noisy homogeneous medium.

ACKNOWLEDGMENT

Charles H. Bennett's experiments with Toom's media and his discussions of the related physics were a source of inspiration. This is by far not a self-correcting manuscript, therefore my great thanks are due to Weiguo Wang and the referee for their careful reading resulting in the correction of many errors. Supported in part by NSF Grant DCR 8603727. Part of the present work was completed with the support of Bell Communications Research, NJ.

REFERENCES

[B] C. H. Bennett, "Logical depth and physical complexity." In R. Herken, editor, *Universal Turing Machie: a Half-Century Survey*, pages 227–257, Oxford Univ. Press, Oxford, 1988.

[Bl] R. E. Blahut, *Theory and Practice of Error-Control Codes*. Addison-Wesley, Reading, MA, 1983.

[BCG] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for Your Mathematical Plays*. Academic Press, London, 1982.

[Bo] A Borodin and I. Munro, *Computational Complexity of Algebraic and Numeric Problems*. American Elsevier, New York, 1975.

[BS] P. Berman and J. Simon, "Investigations of fault-tolerant networks of computers," *Proc. 20th ACM Symp. Th. Computing* (1988).

[DO1] R. L. Dobrushin and S. I. Ortyukov, "Lower bounds for the redundancy of self-correcting arrangements of unreliable functional elements," *Problems Inf. Transmiss.* 13(1): 59–65 (1977).

[DO2] R. L. Dobrushin and S. I. Ortyukov, "Upper bound on the redundancy of self-correcting arrangements of unreliable elements," *Problems Inf. Transmiss.* 13(3): 201–208 (1977).

[G] P. Gács, "Reliable computation with cellular automata," *J. Computer Syst. Sci.* 32(1): 15–78 (1986).

[GR] P. Gács and J. Reif, "A simple three-dimensional real-time reliable cellular array," *J. Computer Syst. Sci.* Vol. 36, No. 2, April 1988, pages 125–147.

[Kur] G. L. Kurdyumov, "An example of a nonergodic homogeneous one-dimensional random medium with positive transition probabilities," *Soviet Math. Dokl.* 19(1): 211–214 (1978).

[Kuz] A. V. Kuznietsov, "Information storage in a memory assembled from unreliable components," *Problems Inf. Transmiss.* 9(3): 254–264 (1973).

[La] S. Lang, *Algebra*. Addison-Wesley, Reading, MA, 1965.

[Le] L. A. Levin, "Randomness conservation inequalities; information and independence in mathematical theories," *Inf. Control* 61(1): 15–37 (1984).

[Li] T. M. Liggett, *Interactive Particle Systems* (monograph). Springer, New York, 1985.

[Ly] N. Lynch, "Easy impossibility proofs for distributed consensus problems," *Distributed Computing* 1(1): 26–39.

[M] N. Margolus, "Physics-like models of computation," *Physica* 10D: 81–85 (1984).

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

[Tof] T. Toffoli and N. Margolus, *Cellular Automata Machines—A New Environment for Modeling*. MIT Press, Cambridge, 1986.

[Too] A. L. Toom, "Stable and attractive trajectories in multicomponent systems," In *Advances in Probability* (Multicomponent Systems), (R. L. Dobrushin, ed.), Vol. 6, pp. 549–575. Dekker, New York, 1980.

[Ts] B. S. Tsirel'son, "Reliable information storage in a system of locally interacting unreliable elements," in *Interacting Markov Processes in Biology*, Lecture Notes in Math., Vol. 653. Springer-Verlag, New York/Berlin, 1978.

[VN] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," In *Automata Studies* (Shannon and McCarthy, eds.). Princeton University Press, Princeton, 1956.

THE COMPLEXITY OF PERFECT ZERO-KNOWLEDGE

Lance Fortnow

ABSTRACT

A *perfect zero-knowledge* interactive proof system convinces a verifier that a string is in a language without revealing any additional knowledge in an information-theoretic sense. We show that for any language that has a perfect zero-knowledge proof system, its complement has a short interactive protocol. This result implies that there are not any perfect zero-knowledge protocols for NP-complete languages unless the polynomial time hierarchy collapses. This chapter demonstrates that knowledge complexity can be used to show that a language is easy to prove.

1. INTRODUCTION

Interactive protocols and zero-knowledge, as described by Goldwasser, Micali, and Rackoff [GMR], have in recent years

Advances in Computing Research, Volume 5, pages 327-343.

ISBN: 0-89232-896-7

proven themselves to be important models of computation in both complexity and cryptography.

Interactive proof systems are a randomized extension to NP that give us a greater understanding of what an infinitely powerful machine can prove to a probabilistic polynomial one. Recent results about interactive protocols have given us an idea of what languages may be efficiently provable in this way.

Zero-knowledge interactive protocols give us a good way to determine which languages can be efficiently proven without giving away any details of the proof. This model consists of an infinitely powerful prover trying to convince a polynomial time verifier that a string is in a certain language. Zero-knowledge requires that the verifier not learn any information useful to him as a polytime machine. Goldreich, Micali, and Wigderson [GMW] show that if one way functions exist then all languages in NP have zero-knowledge proofs. However, their proof relies on the fact that the verifier has limited power and is unable to invert these one-way functions. A stronger notion is that of perfect zero-knowledge (PZK) that requires that the verifier not learn any additional information no matter how powerful he may be. There are many languages not known to be in BPP or NP $\cap$ co-NP, such as graph isomorphism [GMW], which have perfect zero-knowledge proof systems.

Our main theorem says that for any language that has a perfect zero-knowledge protocol, its complement has a single round interactive protocol. Thus PZK $\subseteq$ co-AM, where AM is the class accepted by one-round Arthur–Merlin games as describes by Babai [B]. Our result holds in the weaker case where we only require that the verifier that follows the protocol will not learn any additional information.

Combining our main theorem with a result of Boppana, Hastad, and Zachos [BHZ], we get that NP-complete languages do not have perfect zero-knowledge proof systems unless the polynomial time hierarchy collapses to the second level. Thus it is unlikely that the result of [GMW] that NP has zero-knowledge proof systems will extend to perfect zero-knowledge.

Our proof makes use of an approximate upper bound protocol that is of independent interest and that may be useful in completely different contexts. This is in contrast to an approximate lower bound protocol used in [S, B, GS].

The results in this chapter do not depend on any unproven cryptographic assumptions.

2. NOTATION AND DEFINITIONS

Let P, the prover, be a probabilistic infinite power Turing machine and V, the verifier, be a probabilistic polynomial time machine that share the same input and can communicate with each other. Let $P \leftrightarrow V$ denote the interaction between P and V. $P \leftrightarrow V(x)$ accepts if after the interaction, V accepts. V's *view of the conversation* between P and V consists of all the messages between P and V and the random coin tosses of V.

P and V form an interactive protocol for a language L

1. If $x \in L$ then $\Pr[P \leftrightarrow V(x) \text{ accepts}] \geqslant \frac{2}{3}$.
2. If $x \notin L$ then $\forall P^* \Pr[P^* \leftrightarrow V(x) \text{ accepts}] \leqslant \frac{1}{3}$.

A *round* of an interactive protocol is a message from the verifier to the prover followed by a message from the prover to the verifier. AM is the class of languages accepted by bounded round interactive protocols.

Messages in a conversation will be described by

$$\beta_1, \alpha_1, \beta_2, \ldots, \beta_k, \alpha_k$$

where the α_i are messages from the prover to the verifier at round i and the β_i are messages from the verifier to the prover.

r will be used for the random coin tosses of the verifier.

Formally, we think of P as a function from the input and the conversation so far to a probability distribution of messages. We put no restrictions on the complexity of this function other than requiring the lengths of the messages to be bounded in size by a polynomial in the size of the input. This chapter will use the informal term, *probabilistic infinite power*, to describe the complexity of the prover.

Let IP be the class of all languages that are *efficiently provable*, i.e., accepted by an interactive protocol.

The notation for describing protocols follows:

P: These are computations performed by the prover that can not be seen by the verifier. The prover has probabilistic infinite time to make these computations.

$P \rightarrow V$: This is a message from the prover to the verifier.

V: These are computations performed by the verifier that cannot be seen by the prover. These computations must be performed in probabilistic polynomial time.

$V \to P$: This is a message from the verifier to the prover.

Let M be a simulator for a view of the conversation between P and V. M is a probabilistic polynomial time machine that will output a conversation between P and V including the random coin tosses of V. Thus each run of M will produce

$$r, \beta_1, \alpha_1, \beta_2, \ldots, \beta_k, \alpha_k.$$

Let $P \leftrightarrow V[x]$ denote the distribution of views of conversations between P and V. $M[x]$ denotes the distribution of views of conversations created by running M on x.

Let $A[x]$ and $B[x]$ be two distributions of strings. $A[x]$ and $B[x]$ are *statistically close* if for any subset of strings S,

$$\left| \sum_{y \in S} \Pr_{A[x]}(y) - \sum_{y \in S} \Pr_{B[x]}(y) \right| < \frac{1}{q(|x|)}$$

for all polynomials q with $|x|$ large enough. Let J be a probabilistic polynomial time machine that outputs either 0 or 1. $A[x]$ and $B[x]$ are *polytime indistinguishable* if for any J,

$$|\Pr[J(A[x]) = 1] - \Pr[J(B[x]) = 1]| < \frac{1}{r(|x|)}$$

for all polynomials r with $|x|$ large enough. $J(A[x])$ is the output of J when run on a string chosen from $A[x]$. Note that if $A[x]$ and $B[x]$ are statistically close then they are polytime indistinguishable.

$P \leftrightarrow V$ is *Zero-Knowledge* (ZK) if for any verifier V^* there is a M_{V^*} such that $(\forall x \in L) P \leftrightarrow V^*[x]$ and $M_{V^*}[x]$ are polytime indistinguishable.

$P \leftrightarrow V$ is *Perfect Zero-Knowledge* (PZK) if for any verifier V^* there is a M_{V^*} such that $(\forall x \in L) P \leftrightarrow V^*[x] = M_{V^*}[x]$.

$P \leftrightarrow V$ is *Almost Perfect Zero-Knowledge* (APZK) if for any verifier V^* there is a M_{V^*} such that $(\forall x \in L) P \leftrightarrow V^*[x]$ and $M_{V^*}[x]$ are statistically close.

Interactive protocols and zero-knowledge were introduced in [GMR]. Perfect zero-knowledge was described originally in [GMW].

The class AM was introduced by Babai [B] as the class of languages that have one round interactive protocols where V's message consists exactly of his coin tosses. This was shown to be equivalent to the definition used above by [GS] and [B].

Note that ZK $\supseteq$ APZK $\supseteq$ PZK. The inclusions are not known to be proper but this chapter gives good evidence that ZK $\neq$ APZK.

The results in this chapter require only a weaker version of zero-knowledge: a simulator need exist only for the given P and V and not necessarily for any V^*. For the rest of this chapter we will assume we are in this weaker model and $M = M_V$ is the simulator for P and V.

3. RELATED RESULTS

Our result shows that for any language L with an almost perfect zero-knowledge protocol, there exists a bounded round interactive protocol for its complement $\bar{L}$. We can then apply several earlier results about bounded round interactive protocols.

Goldwasser and Sipser [GS] have shown that for any language that has an interactive protocol in Q rounds, there is an Arthur–Merlin protocol in $Q + 2$ rounds for that language. Arthur–Merlin protocols are similar to interactive protocols except that the verifier's message are just random coin tosses. Babai [B] showed that any bounded round Arthur–Merlin protocol is equivalent to a one round Arthur–Merlin protocol. This is just the class AM. Babai also shows that AM $\subseteq NP^R$ for a random oracle R and also that AM $\subseteq \Pi_2^p$. Sipser pointed out that AM is contained in nonuniform NP.

Boppana, Hastad, and Zachos [BHZ] have shown that if co-NP has bounded round interactive proofs then the whole polynomial time hierarchy wouldd be an AM implying that the polynomial time hierarchy collapses to the second level.

Subsequent to our result, Aiello and Hastad [AH] have shown, using similar techniques, that any almost perfect zero-knowledge protocol can be done by a bounded round interactive protocol. This result is a nice complement to our result that describes the complexity of the complement of perfect zero-knowledge languages. Combining the two results we have that any language with a perfect zero-knowledge proof system is in nonuniform NP $\cap$ co-NP.

Brassard and Crépeau [BC] have shown perfect zero-knowledge for SAT using a different model for interactive protocols where the

prover is a polynomial time machine that knows a satisfying assignment. Our result about perfect zero-knowledge relies on the ability of the prover to have infinite power and thus does not apply to Brassard and Crépeau's model.

4. SHOWING SETS ARE LARGE AND SMALL

In this paper, we will need protocols to show sets are large and small. We do both using Carter–Wegman Universal Hash Functions [CW].

Let $\Sigma = \{0,1\}$. Suppose $S \subseteq \Sigma^N$, $0^N \notin S$. Let F be a random binary $b \times N$ matrix. Let f: $\Sigma^N \to \Sigma^b$ be the function defined by $f(x) = Fx$ using regular matrix multiplication modulo two. We can think of f in terms of linear algebra over the field of two elements. f is distributed evenly over all possible linear functions from n-dimensional space to b-dimensional space.

LEMMA 1 (VECTOR INDEPENDENCE). *Suppose* $x_1, x_2, \ldots, x_k \in \Sigma^N$ *are linearly independent vectors over the field of two elements. Then* $f(x_1), f(x_2), \ldots, f(x_k)$ *are independently and uniformly distributed over* Σ^b.

Proof. Since $x_1, x_2, \ldots, x_k$ are linearly independent, we can extend to a basis. Pick $b_{k+1}, b_{k+2}, \ldots, b_N$ to complete the basis. Let T be the transformation matrix from this new basis to the canonical basis of Σ^N. Then the matrix $B = FT$ describes the function from the new basis to the canonical basis of Σ^b. Since T is an invertible matrix, there is a one-to-one correspondence B and F. Thus B is distributed uniformly over all possible binary $b \times N$ matrices. $f(x_j)$ is just the jth column of B. Thus each $f(x_j)$ is independently distributed over Σ^b. □

If $|S| \gg 2^b$ then f_S is likely to be onto most of Σ^b and most elements of Σ^b will have many preimages.

If $|S| \ll 2^b$ then the range of f_S is a small subset of Σ^b and most elements of $f_S(S)$ will have only one inverse in S.

4.1. Lower Bound Protocol

If S is recognizable in polynomial time we use the following protocol to show S is large:

V: Pick l independent random hash functions $f_1, \ldots, f_l$: $\Sigma^N \to \Sigma^b$ and l^2 points $z_1, \ldots, z_{l^2} \in \Sigma^b$
$V \to P$: $f_1, \ldots, f_l, z_1, \ldots, z_{l^2}$
$P \to V$: x
V: Accept if $x \in S$ and $f_i(x) = z_j$ for some $i, j, 1 \leqslant i \leqslant l$ and $1 \leqslant j \leqslant l^2$.

If S is much smaller than 2^b then it is unlikely for there to be any x such that $f_i(x) = z_j$. However if S is large then there are likely to be many x so a prover should have no trouble exhibiting such an x that V can verify in polynomial time.

LEMMA 2 (LOWER BOUND). [GS] *Using the above protocol with a given* $N, b, d > 0$ *and* $l > \max\{b, 8\}$.

1. *If* $|S| \geqslant 2^b$ *then* $\Pr(P \leftrightarrow V \text{ accepts}) \geqslant 1 - 2^{-l/8}$.
2. *If* $|S| \leqslant 2^b/d$ *then* $\Pr(P^* \leftrightarrow V \text{ accepts}) \leqslant l^3/d$ *for any* P^*.

4.2. Upper Bound Protocol

If V has a random element $s \in S$ that is not known by P then the following protocol is used to show S is small:

V: Pick a random $N \times b$ matrix F
$V \to P$: $F, f(s) = Fs$
$P \to V$: s.

If S is small then s is likely to be the only element of S that maps to $f(s)$; thus P can find s. If S is large then many elements of S map to $f(s)$, and because s is a random element of S, the prover will have no way of determining which element of S V has.

LEMMA 3 (UPPER BOUND). *Using the above protocol with a given* $N, b > 0$ *and* $d > 7$.

1. *If* $|S| \leqslant 2^b/d$ *then* $\Pr(P \leftrightarrow V \text{ accepts}) \geqslant 1 - (1/d)$.
2. *If* $|S| > 8d2^b$ *then* $\Pr(P^* \leftrightarrow V \text{ accepts}) \leqslant (1/d)$ *for any* P^*.

Proof. Let A be the random variable equal to the number of $x \neq s$ in S such that $f(x) = f(s)$. Let $S' = S - \{s\}$. Let A_x be the indicator random variable equal to one if $f(x) = f(s)$, zero

otherwise. Then

$$E(A) = E\left(\sum_{x \in S'} A_x\right) = \sum_{x \in S'} E(A_x) = \sum_{x \in S'} 2^{-b} = \frac{|S| - 1}{2^b}.$$

If $|S| \leqslant 2^b/d$ then $E(A) \leqslant 1/d$. If $f(s)$ has only s as an inverse in $|S|$ then P with his infinite power will be able to determine s. Thus $\Pr(P \leftrightarrow V \text{ rejects}) \leqslant \Pr(A \geqslant 1) \leqslant E(A) \leqslant 1/d$ since A is an integral random variable.

Suppose $|S| > 8d2^b$. We can assume $|S| = 8d2^b + 1$ without increasing the probability of acceptance. Then $E(A) = 8d$. Since P^* has no idea what s V has, P^* can only have a $1/(A+1)$ probability of predicting the s that V has. We will show that there is a high probability that A is large. To show this we look at the variance of A.

Given x, y, s all distinct and $y \neq x \oplus s$ then x, y, s are linearly independent. Then by the Vector Independence Lemma $f(x)$, $f(y)$, $f(s)$ are independently distributed over Σ^b. It then follows that A_x and A_y are independent random variables and their covariance is zero.

The covariance of any two indicator random variables is never greater than the expected value of one of them.

$$\begin{aligned} VAR(A) &= \sum_{x,y \in S'} COV(A_x, A_y) \\ &= \sum_{x \in S'} [COV(A_x, A_x) + COV(A_x, A_{x \oplus s})] \\ &\leqslant \sum_{x \in S'} 2E(A_x) \leqslant 16d. \end{aligned}$$

It is possible that $x \oplus s \notin S$ but this would only decrease the variance. Using Chebyshev's inequality we get

$$\Pr(A < 2d) \leqslant \Pr(|A - 8d| \geqslant 6d) \leqslant \frac{VAR(A)}{36d^2} \leqslant \frac{16d}{36d^2} \leqslant \frac{1}{2d}.$$

So with probability at most $1/2d$ A is small enough that P^* can determine s easily; otherwise P^* has at most $1/2d$ chance of guessing s, so in total P^* has at most a $1/d$ chance of determining s. □

4.3. Comparison Protocol

Suppose we had two sets $S_1, S_2 \subseteq \Sigma^N$ and wanted to show that $|S_1| \gg |S_2|$. If S_1 is polynomial time testable and V has a random element s_2 of S_2, then the following is a protocol to show $|S_1| \gg |S_2|$:

$P \to V$: $b' \leqslant N$
$P \to V$: Use lower bound protocol on S_1 with $b = b', l = 8nN$
$P \to V$: Use upper bound protocol on S_2 with $b = b' - 3n$, $s = s_2$.

LEMMA 4 (COMPARISON). *Using the above protocol*:

1. *If* $|S_1| \geqslant 2^{4n+1}|S_2|$ *then* $\Pr(P \leftrightarrow V \text{ accepts}) \geqslant 1 - 2^{1-n}$.
2. *If* $|S_1| \leqslant 2^{n-4}|S_2|$ *then* $\Pr(P^* \leftrightarrow V \text{ accepts}) \leqslant n^3 N^3 2^{9-n}$ *for any* P^*.

Proof. Let $d = 2^n$.

1. Pick $b' = \lfloor \log |S_1| \rfloor$. Then each protocol accepts with probability $\geqslant 1 - 2^{-n}$, so that both will accept with probability $\geqslant 1 - 2^{1-n}$ by the upper and lower bound lemmas.
2. There are two cases depending on what b' P chooses:
   (a) If $b' \geqslant \lceil \log |S_1| \rceil - n$ then by the lower bound lemma the probability of V accepting is $\leqslant n^3 N^3 2^{9-n}$.
   (b) If $b' < \lceil \log |S_1| \rceil + n$ then by the hypothesis $b' - 3n < \lfloor \log |S_2| \rfloor - n - 3$ and by the upper bound lemma the probability of V accepting is $\leqslant 2^{-n}$. □

Using Carter–Wegman Hashing to show a set is large was introduced by Sipser [S] and used extensively in [S, B, GS]. To the author's knowledge this paper is the first use of an interactive protocol to show a set is small.

5. MAIN THEOREM

We will start with a simple version of the theorem:

THEOREM 1. *Let L be a language with a perfect zero-knowledge interactive protocol. Then there exists an interactive protocol accepting* $\bar{L}$.

5.1. Structure of Proof

We are given a prover and verifier (P and V) for the language L, and a simulator M that produces views of conversations between P and V and the random coin tosses of V. Note that one can simulate the computation of V in polynomial time, checking, for example, whether or not V accepts. On $x \in L$, M produces a view of a conversation for exactly the same probability distribution as when P and V run on x. The key idea of the proof of the theorem is to notice what M might do on $x \notin L$. There is no requirement in the definition of perfect zero-knowledge on what M does on $x \notin L$; however there are three possibilities:

1. M will produce "garbage," something that clearly is not a randomly selected member of $P \leftrightarrow V[x]$.
2. M will produce views of conversations that cause V to reject most of the time.
3. M will produce a simulation that looks valid and causes V to accept. It may not be possible in polynomial time to tell this view from one created by P and V when $x \in L$. However, M must be producing views of conversations from a distribution quite different from the distribution of views between P and V, since in the real views V is likely to reject.

We will create a new prover and verifier, P' and V' that will determine if one of the three cases occur. V' will run M and get a view of a conversation between P and V and r, the random coin tosses of V. V' will check that this view is valid and that V halts accepting. If the view fails this test then it falls in cases 1 or 2 so V' knows that $x \notin L$ and V' accepts. Otherwise V' will send to P' some initial segment of the conversation. P' will then convince V' that the conversation came from a bad distribution by "predicting" r better than P' could have done from a good distribution.

5.2. An Example: Graph Isomorphism

Graph isomorphism is a well-studied problem that is clearly in NP but not known to be in co-NP or BPP. A perfect zero-knowledge proof of graph isomorphism was presented in [GMW]. We will show how our theorem converts this perfect zero-knowledge protocol to an interactive protocol for graph nonisomorphism. This protocol for graph nonisomorphism is identical to the graph

nonisomorphism protocol described in [GMW]; our proof, however, shows that the similarity between the two protocols is not coincidental.

Let π: $\{1,\ldots,n\} \to \{1,\ldots,n\}$ be a permutation of the vertices of a graph. For a graph $G = [V,E]$ let $[\pi(v_1),\pi(v_2)] \in \pi(E) \Leftrightarrow (v_1,v_2) \in E$. Let $\pi(G) = [V,\pi(E)]$.

Two graphs G_1 and G_2 are *isomorphic* if there exists a permutation π such that $\pi(G_1) = G_2$. A perfect zero-knowledge protocol for graph isomorphism suggested by [GMW] works as follows:

P: Generate random permutation π and computes $G = \pi(G_1)$
$P \to V$: G
$V \to P$: $i = 1$ or 2 chosen at random
$P \to V$: π' chosen at random such that $\pi'(G_i) = G$.

If $G_1 \cong G_2$ then G will be a permutation of both G_1 and G_2 and P will always be able to find a π'. If $G_1 \ncong G_2$ then G cannot be a permutation of both G_1 and G_2, so at least half of the time V will choose an i such that no π' exists. Thus we have an interactive protocol for graph isomorphism. This protocol also is perfect zero-knowledge.

The simulator M works as follows:

M generates π and i at random and computes $G = \pi(G_i)$, then outputs the following view of a conversation:

r: i
$P \to V$: G
$V \to P$: i
$P \to V$: π.

It is easy to verify that when $G_1 \cong G_2$, M produces exactly the same distribution of views of conversations as P and V. Notice what happens when $G_1 \ncong G_2$. The output of M always causes V to accept. Thus when $G_1 \ncong G_2$, M must produce views of conversations from a very different distribution from what P and V produce. In fact whenever $G_1 \ncong G_2$, one can always predict $r = i$ from the G produced by M. This leads to a new interactive protocol between a new prover and verifier, P' and V', for graph nonisomorphism as follows:

V': Generate π and i at random and compute $G = \pi(G_i)$
$V' \to P'$: G
$P' \to V'$: i.

5.3. The Protocol for $\bar{L}$

We are given a prover and verifier, P and V for a language L and a simulator M that exactly simulates views of conversations between P and V when $x \in L$. Let $n = |x|$ and let k be the number of rounds of the protocol which is bounded by a polynomial in n. We can decrease the probability of error in the protocol between P and V to 2^{-n^t} for any constant t by the standard trick of running the protocol several times in parallel and having V accept if the majority of individual protocols accept. This new protocol is still perfect zero-knowledge—we just run the simulator in parallel. Note that we make use of the fact that we only need a simulator for the real verifier V. In general, it is not known whether perfect zero-knowledge protocols remain perfect zero-knowledge when run in parallel.

Thus we can assume

1. If $x \in L$ then $\Pr[P \leftrightarrow V(x) \text{ accepts}] \geqslant 1 - 2^{-6kn}$.
2. If $x \notin L$ then $\forall P^* \Pr[P^* \leftrightarrow V(x) \text{ accepts}] \leqslant 2^{-6kn}$.

For the sake of the comparison protocol, let us require that V immediately rejects if all its coin tosses are zero. Since this will happen with an exponentially small probability it will not affect the correctness of the protocol. The protocol remains perfect zero-knowledge by having the simulator output no conversation if the verifier's coins are all zero.

A protocol between a new prover and verifier, P' and V', works as follows:

V': Run M and get $r, \beta_1, \alpha_1, \ldots, \beta_k, \alpha_k$. V' now checks two things:
1. Check that the conversation is *valid*, i.e., that $r, \alpha_1, \ldots, \alpha_k$ will cause V to say $\beta_1, \ldots, \beta_k$.
2. Check that the conversation causes V to accept.

If either of these tests fail then V' can be very sure that $x \notin L$ so V' quits now and accepts. Otherwise V' continues.
Let $j = 1$.

$V' \rightarrow P'$: β_j, α_j

$P' \rightarrow V'$: Look at the sets $\mathscr{R}_1$ and $\mathscr{R}_2$ as defined below. If $|\mathscr{R}_1| \geqslant 2^{4n+1}|\mathscr{R}_2|$ then use the comparison protocol

described in Section 4 to show $|\mathscr{R}_1| \gg |\mathscr{R}_2|$. Otherwise let $j = j + 1$. If $j \leqslant k$ tell V' to TRY NEXT ROUND, otherwise GIVE UP.

$\mathscr{R}_1$ can be thought of as all the possible random strings of V after round j of the protocol. $\mathscr{R}_2$ are the possible random strings of V generated by M. More formally:

Let $\mathscr{R}$ be the set of all possible coin tosses of V.
Let $\mathscr{R}_1 = \{R \in \mathscr{R} \mid R \text{ and } \alpha_1, \ldots, \alpha_j \text{ cause } V \text{ to say } \beta_1, \ldots, \beta_j\}$.
Let $\mathscr{R}_2 = \{R \in \mathscr{R} \mid M \text{ can output } R, \beta_1, \alpha_1, \ldots, \beta_j, \alpha_j \text{ part of a valid, accepting conversation}\}$.

Note that $\mathscr{R}_2 \subseteq \mathscr{R}_1$ and if $x \in L$ then $\mathscr{R}_2 \approx \mathscr{R}_1$. Also note that $\mathscr{R}_1$ is independent of α_j.

$\mathscr{R}_1$ is polynomial time testable. If $x \in L$ then M produces the exact distribution between P and V and thus r is a random element of $\mathscr{R}_2$ which P' does not know. In that case we have fulfilled the requirements of the comparison protocol. If $x \notin L$ it is possible that r is not a random element of $\mathscr{R}_2$ but this can only increase the probability of the comparison protocol accepting.

5.4. The Protocol Constitutes an Interactive Protocol for $\bar{L}$

To show that this is an interactive protocol for $\bar{L}$, we must show two things:

1. If $(x \in \bar{L})$ then $P' \leftrightarrow V'(x)$ accepts with probability $\geqslant \frac{2}{3}$.
2. If $(x \notin \bar{L})$ then $\forall \hat{P}\ \hat{P} \leftrightarrow V'(x)$ accepts with probability $\leqslant \frac{1}{3}$.

We will prove the second statement first since it is the easier of the two to prove.

2. Suppose $x \in L$. Then M will produce views of conversations from exactly the same distribution as P and V. Thus every conversation produced by M will be valid. Assume a prover $\hat{P}$ is able to convince V' to accept with probability $\geqslant \frac{1}{3}$. There may be an exponentially small chance that V will reject in this conversation and this will cause V' to accept. If $|\mathscr{R}_1| \leqslant 2^{n-4}|\mathscr{R}_2|$ on any round then the comparison protocol will accept with a exponentially small probability. Thus we can assume that with probability $> \frac{1}{4}$ that $|\mathscr{R}_1| > 2^{n-4}|\mathscr{R}_2|$ for some round j. Since M outputs all possible conversations, $\mathscr{R}_2$ is just the random coin tosses of V that might

cause V to accept in the future. So at round j of the protocol, the probability of V's acceptance $< |\mathscr{R}_2|/|\mathscr{R}_1| \leqslant 2^{4-n}$. Since this happens at least a fourth of the time the probability of V's acceptance in general is $\leqslant \frac{3}{4} + 2^{4-n}$ that contradicts the fact that V will accept with probability greater than $1 - 2^{-6kn}$.

1. Suppose to the contrary that $x \notin L$ and the protocol does not work. If $|\mathscr{R}_1| < 2^{4n+1}|\mathscr{R}_2|$ then by the comparison lemma the comparison protocol will fail with at most an exponentially small probability. So $|\mathscr{R}_1| \geqslant 2^{4n+1}|\mathscr{R}_2|$ at all rounds j with probability at least one-fourth. We use this to derive a contradiction by demonstrating that $P \leftrightarrow V$ is not an interactive proof system for L by presenting a prover P^* that will convince V (the original verifier) that $x \in L$ with probability greater than 2^{-6kn}.

At round j suppose the conversation so far has been $\beta_1', \alpha_1', \ldots, \beta_j'$. P^* works as follows:

P^*: Run M which outputs $r, \beta_1, \alpha_1, \ldots, \beta_k, \alpha_k$. Check that this is a valid accepting conversation. If not, try again. See if $\beta_1, \alpha_1, \ldots, \beta_j = \beta_1', \alpha_1', \ldots, \beta_j'$. If not, try again.

$P^* \rightarrow V$: α_j.

At round j when P^* has a conversation from M that matches the conversation so far, $\mathscr{R}_1$ is the set of possible random coin tosses of V. When P^* says α_j, $\mathscr{R}_2$ is the set of coin tosses of V that will still keep V heading toward an accepting path. Since $|\mathscr{R}_2| \geqslant 2^{5n+1}|\mathscr{R}_1|$, this will happen with probability $\geqslant 2^{-(5n+1)}$. So after k rounds, V will end up accepting with a probability at least $\frac{1}{4}2^{-(5kn+k)}$, which is higher than the 2^{-6kn} maximum accepting probability we assumed for V and any P^*, when $x \notin L$. □

Note that P^* may require exponential expected time to complete its part of the protocol but in our model an infinitely powerful P^* is allowed.

6. EXTENSIONS AND COROLLARIES

THEOREM 2. *Suppose $P \leftrightarrow V$ is an interactive protocol for a language L and there is a probabilistic polynomial time simulator M such that $M[x]$ is statistically close to $P \leftrightarrow V[x]$. Then there is a single round interactive protocol for the complement of L.*

Idea of proof. This extends the main theorem in two ways. First, we do not require $M[x] = P \leftrightarrow V[x]$, just that they be statistically close. One can check the proof in the previous section and notice that, with some minor adjustments to the probabilities, statistically close is good enough.

Second, we would like to get a single round protocol for the complement of L. Notice that in the protocol given above the number of rounds is dependent on when P' decides to say STOP. To get bounded rounds we must make the following change to the protocol:

V': Run M k^3 times independently and get k^3 views of conversations; check that each conversation is valid and accepting.

$V' \to P'$: For $1 \leqslant i \leqslant k^3$ send the first $i \bmod k$ rounds of the ith conversation.

$P' \to V'$: Pick any conversation j and show $|\mathscr{R}_1| \gg |\mathscr{R}_2|$ for the view of that conversation.

It is not hard to verify that the above proof still works for this new protocol. Once we have bounded rounds we apply the theorems of [B, GS] which imply that all bounded round protocols can be made into single round protocols.

Some trivial corollaries follow from results that are described in Section 3.

COROLLARY 1. *If $L \in APZK$ then $\bar{L} \in AM$.*

COROLLARY 2. *If L has an almost perfect zero-knowledge interactive protocol (possibly with an unbounded number of rounds) then $L \in (NP \cap co\text{-}NP)^R$, where R is a random oracle.*

COROLLARY 3. *If any NP-complete language has an almost perfect zero-knowledge interactive protocol then the polynomial time hierarchy collapses to the second level.*

COROLLARY 4. *If there are one-way functions and the polynomial time hierarchy does not collapse then $NP \subseteq ZK$; but $NP \not\subseteq APZK$, so $ZK \neq APZK$.*

7. OPEN PROBLEMS

There are several interesting problems remaining, including

- What is the relationship between PZK and APZK?
- Are complement of perfect or almost perfect zero-knowledge languages themselves perfect zero-knowledge in any sense?
- Are cryptographic assumptions necessary to show NP has zero-knowledge protocols? Although this chapter shows that NP probably does not have perfect zero-knowledge proof systems, it is still conceivable that the intractability of SAT is a good enough assumption for a zero-knowledge protocol.

ACKNOWLEDGMENTS

The author would like to express his gratitude to his advisor, Mike Sipser, for his support and encouragement.

The author would also like to thank Mike, Silvio Micali, Oded Goldreich, Joan Feigenbaum, Paul Beame, Eric Schwabe, and Su-Ming Wu for their useful comments on this paper.

The author is supported in part by an Office of Naval Research fellowship, NSF Grant DCR-8602062, and Air Force Grant AFOSR-86-0078. Much of this work was done while the author was at the University of California at Berkeley.

REFERENCES

[AH] W. Aiello and J. Hastad, "Statistical zero-knowledge languages can be recognized in two rounds," *JCSS*, to appear. Extended Abstract available in *Proc. 28th FOCS* 439–448 (1987).

[B] L. Babai, "Trading group theory for randomness," *Proc. 17th STOC* 421–429 (1985).

[BHZ] R. Boppana, J. Hastad, and S. Zachos, "Does co-NP have short interactive proofs," *IPL* 25(2): 127–132 (1987).

[BC] G. Brassard and C. Crépeau, "Non-transitive transfer of confidence: A perfect zero-knowledge interactive protocol for SAT and beyond," *Proc. 27th FOCS* 188–195 (1986).

[CW] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *JCSS* 18(2): 143–154 (1979).

[GMW] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design," *Proc. 27th FOCS* 174–187 (1986).

[GMR] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," *SIAM Journal on Computing* 18(1): 186–208 (1989). Extended Abstract available in *Proc. 17th STOC* 291–304 (1985).

[GS] S. Goldwasser and M. Sipser, "Private coins versus public coins in interactive proof systems," In *Randomness and Computation*, Vol. 5 of *Advances in Computing Research* (S. Micali, ed.). JAI Press, Greenwich, 1987. Extended Abstract available in *Proc. 18th STOC* 59–68 (1986).

[S] M. Sipser, "A complexity theoretic approach to randomness," *Proc. 15th STOC* 330–335 (1983).

RANDOMIZED ROUTING ON FAT-TREES

Ronald I. Greenberg and Charles E. Leiserson

ABSTRACT

Fat-trees are a class of routing networks for hardware-efficient parallel computation. This chapter presents a randomized algorithm for routing messages on a fat-tree. The quality of the algorithm is measured in terms of the *load factor* of a set of messages to be routed, which is a lower bound on the time required to deliver the messages. We show that if a set of messages has load factor λ on a fat-tree with n processors, the number of delivery cycles (routing attempts) that the algorithm requires is $O(\lambda + \lg n \lg\lg n)$ with probability $1 - O(1/n)$. The best previous bound was $O(\lambda \lg n)$ for the off-line problem in which the set of messages is known in advance. In the context of a VLSI model that equates hardware cost with physical volume, the routing algorithm can be used to demonstrate that fat-trees are universal routing networks. Specifically, we

Advances in Computing Research, Volume 5, pages 345–374.

ISBN: 0-89232-896-7

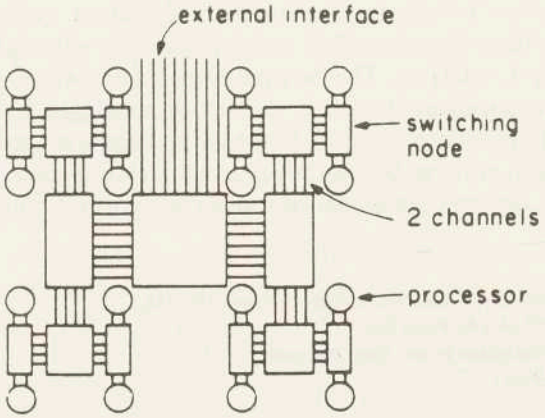
prove that any routing network can be efficiently simulated by a fat-tree of comparable hardware cost.

1. INTRODUCTION

Fat-trees constitute a class of routing networks for general-purpose parallel computation. This paper presents a randomized algorithm for routing a set of messages on a fat-tree. The routing algorithm and its analysis generalize an earlier *universality* result by showing, in a three-dimensional VLSI model, that any network can be efficiently simulated by a fat-tree of comparable volume. The result had been proved only for *off-line* simulations [Le2], where the communication pattern is known in advance; this chapter extends it to the more interesting *on-line* case, where messages are spontaneously generated by processors.

As is illustrated in Figure 1, a fat-tree is a routing network based on Leighton's tree-of-meshes graph [L]. A set of n processors is located at the leaves of a complete binary tree. Each edge of the underlying tree corresponds to two *channels* of the fat-tree: one from parent to child, the other from child to parent. Unlike a normal tree that is "skinny all over," in a fat-tree, each channel consists of a bundle of wires. The number of wires in a channel c is called its capacity, denoted by $\text{cap}(c)$. Each internal node of the fat-tree contains circuitry that switches messages from incoming to outgoing channels.

Figure 1. The organization of a fat-tree. Processors are located at the leaves, and the internal nodes contain concentrator switches. The channels between internal nodes consist of bundles of wires.



The channel capacities of a fat-tree determine the amount of hardware required to build it. The greater the capacities of the channels, the greater the communication potential, and also, the greater the hardware cost of an implementation of the network. The idea of fat-trees is to take advantage of a principle of locality in much the same way as does the telephone network: by using only slightly more hardware than that required to support fast *nonlocal* communication among a set of processors, much additional *local* communication among a larger set of processors can be supported.

An issue that any routing algorithm for a fat-tree must face is that some communication patterns among the processors are harder than others. For example, suppose the channel capacity between the two halves of a fat-tree is $\Theta(n^{2/3})$, where n is the number of processors, and suppose each processor sends a message to a processor in the other half. Since the number of messages that must pass through the root is n and the capacity is $\Theta(n^{2/3})$, the time required by the network to deliver all the messages is $\Omega(n^{1/3})$ because of congestion. In contrast, there are many communication patterns among n processors with less nonlocal communication that can be implemented in subpolynomial time.

The routing algorithm for fat-trees presented in this chapter is a randomized algorithm that we analyze in terms of a measure of congestion called the *load factor*. The load factor of a set of messages is the largest ratio over all channels in the fat-tree of the number of messages that must pass through the channel divided by the capacity of the channel. The load factor of a set of messages is thus a lower bound on the time to deliver the messages.

We show in this chapter that our routing algorithm can deliver a set of messages with load factor λ in $O(\lambda + \lg n \lg\lg n)$ delivery cycles (routing attempts) with high probability. The best previous bound for a problem of this nature was an $O(\lambda \lg n)$ bound for the off-line situation where the set of messages is known in advance [Le2] and the problem is to schedule their delivery. The analysis of our randomized routing algorithm makes no assumptions about the statistical distribution of messages, except insofar as it affects the load factor. Moreover, the algorithm is not restricted to permutation routing or situations where each processor can send or receive only a constant number of messages, as is common in the literature. We consider the general situation where each processor can send and receive polynomially many messages.

Our routing algorithm also differs from others in the literature in the way randomization is used. Unlike the algorithms of Valiant [V], Valiant and Brebner [VB], Aleliunas [Al], Upfal [U], and Pippenger [P], for example, it does not randomize with respect to paths taken by messages. For example, Valiant's classic scheme for routing on a hypercube sends each message to a randomly chosen intermediate destination and, from there, to its true destination. On a fat-tree, such a technique would likely convert communication patterns with good locality into ones with much global communication. Instead of choosing random paths for messages to traverse, our algorithm repeatedly attempts to deliver a randomly chosen subset of the messages. A by-product of this strategy is that our algorithm requires no intermediate buffering of messages.

The remainder of this chapter is organized as follows. Section 2 describes fat-trees in more detail. Section 3 presents the randomized algorithm for efficiently routing messages on the fat-tree network, and Section 4 contains the full analysis of the algorithm. Section 5 proves that fat-trees are universal in VLSI models. Specifically, we use the randomized routing algorithm to show that a fat-tree with properly chosen channel capacities can efficiently simulate any other network of comparable hardware cost, where cost is measured as the area or volume of the circuitry. Section 6 gives an existential lower bound for a class of naive greedy routing algorithms that shows that the greedy strategy is inferior to our randomized algorithm for worst case inputs. Section 7 contains several additional results. These include a modification of the routing algorithm that achieves better bounds when each channel has capacity $\Omega(\lg n)$, a new, simpler fat-tree design, and results on off-line routing. Finally, Section 8 contains some concluding remarks.

2. FAT-TREES

This section describes fat-trees in more detail. We give more specific implementation details on our routing strategy and the hardware required to support it. We precisely define the *load factor* of a set of messages on a general network in terms of cuts of the network, and we prove that it suffices in a fat-tree to consider only the load factors on channels.

The implementation of fat-trees described here follows that of [Le2]. We consider communication through the fat-tree network to

be synchronous, bit serial, and batched. By synchronous, we mean that the system is globally clocked. By bit serial, we mean that the messages can be thought of as bit streams. Each message snakes its way through the wires and switches of the fat-tree, with leading bits of the messages setting switches and establishing a path for the remainder to follow. By batched, we mean the messages are grouped into *delivery cycles*. During a delivery cycle, the processors send messages through the network. Each message attempts to establish a path from its source to its destination. Since some messages may be unable to establish connections during a delivery cycle, each successfully delivered message is acknowledged through its communication path at the end of the cycle. Rather than buffering undelivered messages, we simply allow them to try again in a subsequent delivery cycle. The routing algorithm is responsible for grouping the messages into delivery cycles so that all the messages are delivered in as few cycles as possible.

The mechanics of routing messages in a fat-tree are similar to routing in an ordinary tree. For each message, there is a unique path from its source processor to its destination processor in the underlying complete binary tree, which can be specified by a relative address consisting of at most $2 \lg n$ bits telling whether the message turns left or right at each internal node. Within each node of the fat-tree, the messages destined for a given output channel are *concentrated* onto the available wires of that channel. This concentration may result in "lost" messages if the number of messages destined for the output channel exceeds the capacity of the channel. We assume, however, that the concentrators within the node are ideal in the sense that no messages are lost if the number of messages destined for a channel is less than or equal to the capacity of the channel. Such a concentrator can be built, for example, with a logarithmic-depth sorting network [Aj]. A somewhat more practical logarithmic-depth circuit can be built by combining a parallel prefix circuit [LF] with a butterfly (i.e., FFT, Omega) network. With switches of logarithmic depth, the time to run each delivery cycle is $O(\lg^2 n)$ bit times, making the natural assumption that messages are $O(\lg n)$ bits long.[1] (Section 7 contains another fat-tree design where the time to run a delivery cycle is $O(\lg n)$ bit times.)

The performance of any routing algorithm for a fat-tree depends on the locality of communication inherent in a set of messages. The locality of communication for a message set M can be summarized

by a measure $\lambda(M)$ called the *load factor*, which we define in a more general network setting.

DEFINITION. Let R be a routing network. A set S of wires in R is a (directed) *cut* if it partitions the network into two sets of processors A and B such that every path from a processor in A to a processor in B contains a wire in S. The *capacity* $\text{cap}(S)$ is the number of wires in the cut. For a set of messages M, define the *load* $\text{load}(M, S)$ of M on a cut S to be the number of messages in M from a processor in A to a processor in B. The *load factor* of M on S is

$$\lambda(M, S) = \frac{\text{load}(M, S)}{\text{cap}(S)},$$

and the *load factor* of M on the entire network R is

$$\lambda(M) = \max_S \lambda(M, S).$$

The load factor of a set of messages on a given network provides a lower bound on the time required to deliver all messages in the set.

For fat-trees, only cuts corresponding to channels need be considered to deteremine the load factor, as is shown by the following lemma.

LEMMA 1. *The load factor of a set M of messages on a fat-tree is*

$$\lambda(M) = \max_c \lambda(M, c),$$

where c ranges over all channels of the fat-tree.

Proof. Any cut S must entirely contain at least one channel. Let us partition the wires in S into $S = c_1 \cup \cdots \cup c_l \cup w$, where $c_1, \ldots, c_l$ are the complete channels in S and w is the set of remaining wires in S. For convenience, let $x_i = \text{load}(M, c_i)$ and $y_i = \text{cap}(c_i)$. Assume without loss of generality that $\lambda(M, c_1) \geqslant \lambda(M, c_i)$ for $i = 1, \ldots, l$, which implies $x_1 y_i - x_i y_1 \geqslant 0$ for all i. The

load factor of M on S is therefore

$$\begin{aligned}\lambda(M, S) &= \frac{x_1 + \cdots + x_1}{y_1 + \cdots + y_l + |w|} \\ &= \frac{x_1}{y_1} - \frac{(x_1 y_2 - x_2 y_1) + \cdots + (x_1 y_l - x_l y_1) + (x_1 |w|)}{y_1 (y_1 + \cdots + y_l + |w|)} \qquad (1) \\ &\leqslant \frac{x_1}{y_1} \\ &= \lambda(M, c_1)\end{aligned}$$

since each term in the numerator of the second term of (1) is nonnegative. □

3. THE ROUTING ALGORITHM

This section gives our randomized algorithm for routing a set M of messages on a fat-tree. The algorithm *RANDOM*, which is based on routing random subsets of the messages in M, is shown in Figure 2. It uses the subroutine *TRY-GUESS* shown in Figure 3. Section 4 provides a proof that on an n-processor fat-tree, the

Figure 2. The randomized algorithm *RANDOM* for delivering a message set M on a fat-tree with n processors. This algorithm achieves the running times in Figure 4 with high probability if the constants k_1 and k_2 are appropriately chosen. Since the load factor $\lambda(M)$ is not known in advance, *RANDOM* makes guesses, each one being tied out by the subroutine *TRY-GUESS*.

```
     Algorithm RANDOM
1    send M
2    U ← M − {messages delivered}
3    λ_guess ← 2
4    while k1 λ_guess < k2 lg n and U ≠ ∅ do
5        TRY-GUESS(λ_guess)
6        λ_guess ← λ²_guess
7    endwhile
8    λ_guess ← (k2/k1) lg n lg lg n
9    while U ≠ ∅ do
10       TRY-GUESS(λ_guess)
11       λ_guess ← 2λ_guess
12   endwhile
```

Figure 3. The subroutine *TRY-GUESS* used by the algorithm *RANDOM* that tries to deliver the set U of currently undelivered messages. When $\lambda_{guess} \geq \lambda(U)$, this attempt will be successful with high probability, if the constants k_1 and k_2 are appropriately chosen. (The value r is the congestion parameter of the fat-tree defined in Section 4, which is typically a small constant.) In that case, λ is always an upper bound on $\lambda(U)$, which is at least halved in each iteration of the while loop. When the loop is finished, $\lambda(U) \leq 1$, so all the remaining messages can be sent.

```
      procedure TRY-GUESS(λ_guess)
1     λ ← λ_guess
2     while λ > 1 do
3         for i ← 1 to max{k1λ, k2 lg n} do
4             independently send each message of U with probability 1/rλ
5             U ← U − {messages delivered}
6         endfor
7         λ ← λ/2
8     endwhile
9     send U
10    U ← U − {messages delivered}
```

probability is at least $1 - O(1/n)$ that *RANDOM* delivers all messages in M within $O(\lambda(M) + \lg n \lg\lg n)$ delivery cycles, if the two constants k_1 and k_2 appearing in the algorithm are properly chosen.

The basic idea of *RANDOM* is to pick a random subset of messages to send in each delivery cycle by independently choosing each message with some probability p. This type of message set merits a formal definition.

DEFINITION. A *p-subset* of M is a subset of M formed by independently choosing each message of M with probability p.

We will show in Section 4 that if p is sufficiently small, many of the messages in a p-subset are delivered because they encounter no congestion during routing. On the other hand, if p is too small, few messages are sent. *RANDOM* varies the probability p from cycle to cycle, seeking random subsets of M that contain a substantial portion of the messages in M, but that do not cause congestion.

The algorithm *RANDOM* varies the probability p because the load factor $\lambda(M)$ is not known. The overall structure of *RANDOM* is to guess the load factor and call the subroutine *TRY-GUESS* for each one. The subroutine *TRY-GUESS* determines the probability p based on *RANDOM*'s guess λ_{guess} and a parameter r, called the

congestion parameter of the fat-tree, which is independent of the message set and which will be defined in Section 4. If λ_{guess} is an upper bound on the true load factor $\lambda(M)$, then with high probability, each iteration of the **while** loop in *TRY-GUESS* halves the upper bound on the load factor $\lambda(U)$ of the set U of undelivered messages, as will be shown in Section 4. When the loop is finished, we have $\lambda(U) \leqslant 1$, and all the remaining messages can be delivered in one cycle. The number of delivery cycles performed by *TRY-GUESS* is $O(\lg \lambda_{guess} \lg n)$ if $2 \leqslant \lambda_{guess} \leqslant \Theta(\lg n)$, and the number of cycles is $O(\lambda_{guess} + \lg n \lg \lg n)$ if $\lambda_{guess} = \Omega(\lg n)$.

RANDOM must make judicious guesses for the load factor because *TRY-GUESS* may not be effective if the guess is smaller than the true load factor. Conversely, if the guess is too large, too many delivery cycles will be performed. Since the amount of work done by *TRY-GUESS* grows as $\lg \lambda_{guess}$ when λ_{guess} is small, and as λ_{guess} when λ_{guess} is large, there are two main phases to *RANDOM*'s guessing. (These phases follow the handling of very small load factors, i.e., $\lambda(M) \leqslant 2$.)

In the first phase, the guesses are squared from one trial to the next. Once λ_{guess} is sufficiently large, we move into the second phase, and the guesses are doubled from one trial to the next. In each phase, the number of delivery cycles run by *TRY-GUESS* from one call to the next forms a geometric series. Thus, the work done in any call to *TRY-GUESS* is only a constant factor times all the work done prior to the call. With this guessing strategy, we can deliver a message set using only a constant factor more delivery cycles than would be required if we knew the load factor in advance.

4. ANALYSIS OF THE ROUTING ALGORITHM

This section contains the analysis of *RANDOM*, the routing algorithm for fat-trees presented in Section 3. We shall show that the probability is $1 - O(1/n)$ that *RANDOM* delivers a set M of messages on a universal fat-tree with n processors in $O(\lambda(M) + \lg n \lg \lg n)$ delivery cycles. Figure 4 gives the tighter bounds that we actually prove.

We begin by stating two technical lemmas concerning basic probability. One is a combinatorial bound on the tail of the binomial distribution of the kind attributed to Chernoff [C], and the other is a general, but weak, bound on the probability that a random variable takes on values smaller than the expectation.

Figure 4. The number of delivery cycles required to deliver a message set M on a fat-tree with n processors. All bounds are achieved with probability $1 - O(1/n)$. The bounds on the number of delivery cycles can be summarized as $O(\lambda(M) + \lg n \lg\lg n)$.

load factor	delivery cycles
$0 \leq \lambda(M) \leq 1$	1
$1 \leq \lambda(M) \leq 2$	$O(\lg n)$
$2 \leq \lambda(M) \leq \lg n \lg\lg n$	$O(\lg n \lg(\lambda(M)))$
$\lg n \lg\lg n \leq \lambda(M) \leq n^{O(1)}$	$O(\lambda(M))$

The first lemma is the Chernoff bound. Consider t independent Bernoulli trials, each with probability p of success. It is well known [F] that the probability that there are at least s successes out of the t trials is

$$B(s,t,p) = \sum_{k=s}^{t} \binom{t}{k} p^k (1-p)^{t-k}.$$

The lemma bounds the probability that the number of successes is larger than the expectation pt.

LEMMA 2.

$$B(s,t,p) \leqslant \left(\frac{ept}{s}\right)^s.$$

Proof. The lemma follows from [V, p. 354]. □

The second technical lemma bounds the probability that a bounded random variable takes on values smaller than the expectation.

LEMMA 3. *Let $X \leqslant b$ be a random variable with expectation μ. Then for any $w < \mu$, we have*

$$\Pr\{X \leqslant w\} \leqslant 1 - \frac{\mu - w}{b - w}.$$

Proof. The definition of expectation gives us

$$\mu \leqslant w \Pr\{X \leqslant w\} + b(1 - \Pr\{X \leqslant w\}),$$

from which the lemma follows. □

We now analyze the routing of a p-subset M' of a set M of messages. If the number $\text{load}(M', c)$ of messages in M' that must pass through c is no more than the capacity $\text{cap}(c)$, then no messages are lost by concentrating the messages into c. We shall say that c is *congested by* M' if $\text{load}(M', c) > \text{cap}(c)$. The next lemma shows that the likelihood of channel congestion decreases exponentially with channel capacity if the probability of choosing a given message in M is sufficiently small.

LEMMA 4. *Let M be a set of messages on a fat-tree, let $\lambda(M)$ be the load factor on the fat-tree due to M, let M' be a p-subset of messages from M, and let c be a channel through which a given message $m \in M'$ must pass. Then the probability is at most $(ep\lambda(M))^{\text{cap}(c)}$ that channel c is congested by M'.*

Proof. Channel c is congested by M' if $\text{load}(M', c) > \text{cap}(c)$. There is already one message from the set M' going through channel c, so we must determine a bound on the probability that at least $\text{cap}(c)$ other messages go through c. Using Lemma 2 with $s = \text{cap}(c)$ and $t = \text{load}(M, c)$, the probability that the number of messages sent through channel c is greater than the capacity $\text{cap}(c)$ is less than

$$B(\text{cap}(c), \text{load}(M, c), p) \leqslant \left(\frac{ep\,\text{load}(M, c)}{\text{cap}(c)} \right)^{\text{cap}(c)}$$

$$\leqslant (ep\lambda(M))^{\text{cap}(c)}. \qquad \square$$

The next lemma will analyze the probability that a given message of a p-subset of M gets delivered. In order to do the analysis, however, we must select p small enough so that it is likely that the message passes exclusively through uncongested channels. The choice of p depends on the capacities of channels in the fat-tree. For convenience, we define a parameter of the capacities that will enable us to choose a suitable upper bound for p.

DEFINITION. The *congestion parameter* of a fat-tree is the smallest positive value r such that for each simple path $c_1, c_2, \ldots, c_l$ of channels in the fat-tree, we have

$$\sum_{k=1}^{l} \left(\frac{e}{r} \right)^{\text{cap}(c_k)} \leqslant \frac{1}{2}.$$

The congestion parameter is generally quite small. For any fat-tree based on a complete binary tree, the longest simple path is at most $2 \lg n$, where n is the number of processors, and thus we have $r \leqslant 4e \lg n$. For universal fat-trees (discussed in Section 5), the congestion parameter is a constant because the capacities of channels grow exponentially as we go up the tree. (All we really need is arithmetic growth in the channel capacities.) The congestion parameter is also constant for any fat-tree based on a complete binary tree if all the channels have capacity $\Omega(\lg \lg n)$. Our analysis of *RANDOM* treats the congestion parameter r as a constant, but the analysis does not change substantially for other cases.

We now present the lemma that analyzes the probability that a given message gets delivered.

LEMMA 5. *Let M be a set of messages on a fat-tree that has congestion parameter r, let $\lambda(M)$ be the load factor on the fat-tree due to M, and let m be an arbitrary message in M. Suppose M' is a p-subset of M, where $p \leqslant 1/r\lambda(M)$. Then if M' is sent, the probability that m gets delivered is at least $\frac{1}{2}p$.*

Proof. The probability that $m \in M$ is delivered is at least the probability that $m \in M'$ times the probability that m passes exclusively through uncongested channels. The probability that $m \in M'$ is p, and thus we need only show that, given $m \in M'$, the probability is at least $\frac{1}{2}$ that every channel through which m must pass is uncongested. Let $c_1, c_2, \ldots, c_l$ be the channels in the fat-tree through which m must pass. The probability that channel c_k is congested is less than $(e/r)^{\mathrm{cap}(c_k)}$ by Lemma 4. The probability that at least one of the channels is congested is, therefore, less than

$$\sum_{k=1}^{l} \left(\frac{e}{r}\right)^{\mathrm{cap}(c_k)} \leqslant \frac{1}{2},$$

by definition of the congestion parameter. Thus, the probability that none of the channels are congested is at least $\frac{1}{2}$. □

We now focus our attention on *RANDOM* itself. The next lemma analyzes the innermost loop (lines 3–6) of *RANDOM*'s subroutine *TRY-GUESS*. At this point in the algorithm, there is a set U of undelivered messages and a value for λ. The lemma shows that if λ

is indeed an upper bound on the load factor $\lambda(U)$ of the undelivered messages when the loop begins, then $\lambda/2$ is an upper bound after the loop terminates. This lemma is the crucial step in showing that *RANDOM* works.

LEMMA 6. *Let U be a set of messages on an n-processor fat-tree with congestion parameter r, and assume $\lambda(U) \leqslant \lambda$. Then after lines* 3–6 *of RANDOM's subroutine TRY-GUESS, the probability is at most $O(1/n^2)$ that $\lambda(U) > \frac{1}{2}\lambda$.*

Proof. The idea is to show that the load factor of an arbitrary channel c remains larger than $\frac{1}{2}\lambda$ with probability $O(1/n^3)$. Since the channel c is chosen arbitrarily out of the $4n - 2$ channels in the fat-tree, the probability is at most $O(1/n^2)$ that any of the channels is left with a load factor larger than $\frac{1}{2}\lambda$.

For convenience, let C be the subset of messages that must pass through channel c and are undelivered at the beginning of the innermost loop in *RANDOM*. Let $C_0 = C$, and for $i \geqslant 1$, let $C_i \subseteq C_{i-1}$ denote the set of undelivered messages at the end of the ith iteration of the loop. Notice that we have $\lambda(C_i, c) = |C_i|/\text{cap}(c)$, since we have $|C_i| = \text{load}(C_i, c)$ by definition.

We now show there exist values for the constants k_1 and k_2 in line 3 of *TRY-GUESS* such that for $z = \max\{k_1\lambda, k_2 \lg n\}$, the probability is $O(1/n^2)$ that $\lambda(C_z, c) > \frac{1}{2}\lambda$, or equivalently, that

$$|C_z| > \tfrac{1}{2}\lambda\text{cap}(c). \tag{2}$$

It suffices to prove that the probability is $O(1/n^3)$ that fewer than $\frac{1}{2}|C|$ messages from C are delivered during the z cycles under the assumption that $|C_i| > \frac{1}{2}\lambda\text{cap}(c)$ for $i = 0, 1, \ldots, z - 1$. The intuition behind the assumption $|C_i| > \frac{1}{2}\lambda\text{cap}(c)$ is that otherwise, the load factor on channel c is already at most $\frac{1}{2}\lambda$ at this step of the iteration. The reason we need only bound the probability that fewer than $\frac{1}{2}|C|$ messages are delivered during the z cycles is that inequality (2) implies that the number of messages delivered is fewer than $|C| - \frac{1}{2}\lambda\text{cap}(c) \leqslant |C| - \frac{1}{2}\lambda(C, c)\text{cap}(c) \leqslant \frac{1}{2}|C|$.

We shall establish the $O(1/n^3)$ bound on the probability that at most $\frac{1}{2}|C|$ messages are delivered in two steps. For convenience, we shall call a cycle *good* if at least $\text{cap}(c)/8r$ messages are delivered, and *bad* otherwise. In the first step, we bound the probability that a given cycle is bad. The expected number of messages delivered

in any given cycle is the product of the number of messages that remain to be delivered and the probability that any of these messages is successfully delivered. Using Lemma 5 with $p = 1/r\lambda \leqslant 1/r\lambda(U) \leqslant 1/r\lambda(C_i)$ in conjunction with the assumption that $|C_i| > \frac{1}{2}\lambda\text{cap}(c)$, we can conclude that the expected number of messages delivered in any given cycle is greater than $(1/2r\lambda)\frac{1}{2}\lambda\text{cap}(c) = \text{cap}(c)/4r$. Then by Lemma 3, the probability that a given cycle is bad is at most $1 - 1/(8r - 1) < 1 - 1/8r$.[2]

The second step bounds the probability that a substantial fraction of the z delivery cycles are bad. Specifically, we show that the probability is $1 - O(1/n^3)$ that at least some small constant fraction q of the z cycles are good. By picking $k_1 = 4r/q$, which implies $z \geqslant 4r\lambda/q$, at least $qz\text{cap}(c)/8r \geqslant \frac{1}{2}|C|$ messages are delivered.

We bound the probability that at least $(1 - q)z$ of the z cycles are bad by using a counting argument. There are $\binom{z}{(1-q)z}$ ways of picking the bad cycles, and the probability that a cycle is bad is at most $1 - 1/8r$. Thus, the probability that at most $\frac{1}{2}|C|$ messages are delivered is

$$\Pr\{\leqslant \tfrac{1}{2}|C| \text{ messages delivered}\} \leqslant \binom{z}{(1-q)z}\left(1 - \frac{1}{8r}\right)^{(1-q)z}$$

$$\leqslant (q^q(1-q)^{1-q})^{-z}\left(1 - \frac{1}{8r}\right)^{(1-q)z} \tag{3}$$

$$\leqslant 2^{-z/12r}, \tag{4}$$

where (3) follows from Stirling's approximation (for sufficiently large z), and (4) follows from choosing $q = 1/100r \ln r$ and performing algebraic manipulations. Since $z = \max\{k_1\lambda, k_2 \lg n\}$, if we choose $k_2 = 36r$, the probability that fewer than $\frac{1}{2}|C|$ messages are delivered is at most $1/n^3$. □

Now we can analyze *RANDOM* as a whole.

THEOREM 7. *For any message set M on an n-processor fat-tree, the probability is at least $1 - O(1/n)$ that RANDOM delivers all the messages of M within the number of delivery cycles specified by Figure 4.*

Proof. First, we will show that if $\lambda_{guess} \geqslant \lambda(M)$, the probability is at most $O(1/n)$ that the loop in lines 2 through 8 of *TRY-GUESS* fails to yield $\lambda(U) \leqslant 1$. Initially, $\lambda \geqslant \lambda(U)$, and we know from Lemma 6 that the probability is at most $O(1/n^2)$ that any given iteration of the loop fails to restore this condition as λ is halved. Since there are $\lg \lambda_{guess}$ iterations of the loop, we need only make the reasonable assumption that λ_{guess} is polynomial in n to obtain a probability of at most $O(1/n)$ that $\lambda(U)$ remains greater than 1 after all the iterations of the loop. That this assumption holds can be verified for each of the cases below by noting that $\lambda(M)$ is at most polynomial in n and that λ_{guess} is never much larger than $\lambda(M)$.

Now we just need to count the number of delivery cycles that have been completed by the time we call *TRY-GUESS* with a λ_{guess} such that $\lambda(M) \leqslant \lambda_{guess}$. Let us denote by λ^*_{guess} the first λ_{guess} that satisfies this condition, and then break the analysis down into cases according to the value of $\lambda(M)$.

For $\lambda(M) \leqslant 1$, we do not actually even call *TRY-GUESS*. We need only count the one delivery cycle executed in line 1 of *RANDOM*.

For $1 \leqslant \lambda(M) \leqslant 2$, we need add only the $k_2 \lg n$ cycles executed when we call *TRY-GUESS*(2).

For $2 < \lambda(M) < (k_2/k_1) \lg n$, the number of delivery cycles involved in each execution of *TRY-GUESS* is $O(\lg \lambda_{guess} k_2 \lg n)$, since we perform $O(\lg \lambda_{guess})$ iterations of the loop in lines 2–8 of *TRY-GUESS*, each containing $k_2 \lg n$ iterations of the loop in lines 3–6. The value of λ^*_{guess} is at most $(\lambda(M))^2$, so the number of delivery cycles is $O(\lg n \lg (\lambda(M))^2)$ for the last guess, $O(\lg n \lg \lambda(M))$ for the second-to-last guess, $O(\lg n \lg \sqrt{\lambda(M)})$ for the third-to-last guess, and so on. The total number of delivery cycles is, therefore,

$$\begin{aligned} & \sum_{0 \leqslant i \leqslant 1 + \lg \lg \lambda(M)} O(\lg n \lg (\lambda(M))^{2^{1-i}}) \\ &= \sum_{0 \leqslant i \leqslant 1 + \lg \lg \lambda(M)} O(2^{1-i} \lg n \lg (\lambda(M))) \\ &= O(\lg n \lg \lambda(M)), \end{aligned}$$

since the series is geometric.

for $\lambda(M) > (k_2/k_1) \lg n$, the number of delivery cycles executed by the time we reach line 8 of *RANDOM* is $O(\lg n \lg \lg n)$ according to the preceding analysis, and then we must continue in the quest to reach λ^*_{guess}. If $\lambda(M) \leqslant (k_2/k_1) \lg n \lg \lg n$, then we need only add

the $O(\lg n \lg\lg n) = O(\lg n \lg \lambda(M))$ delivery cycles involved in the single call $TRY\text{-}GUESS((k_2/k_1)\lg n \lg\lg n)$.

If $\lambda(M) > (k_2/k_1)\lg n \lg\lg n$, the number of delivery cycles executed before reaching line 8 is $O(\lg n \lg\lg n)$ as before, which is $O(\lambda(M))$. We must then add $O(\lambda_{guess})$ cycles for each call of *TRY-GUESS* in line 10. Since λ^*_{guess} is at most $2\lambda(M)$, the total additional number of delivery cycles is

$$\sum_{0 \leq i \leq t} O(2^{1-i}\lambda(M)) = O(\lambda(M)),$$

where $t = 1 + \lg(k_1\lambda(M)/k_2 \lg n \lg\lg n)$. The total number of delivery cycles is thus $O(\lambda(M))$.

The $1 - O(1/n)$ bound on the probability that *RANDOM* delivers all the messages can be improved to $1 - O(1/n^k)$ for any constant k by choosing $k_2 = 12(k+2)r$, or by simply running the algorithm through more choices of λ_{guess}.

We can also use *RANDOM* to obtain a routing algorithm that guarantees to deliver all the messages in finite time, and whose expected number of delivery cycles is as given in Figure 4. We simply interleave *RANDOM* with any routing strategy that guarantees to deliver at least one message in each delivery cycle. If the number of messages is bounded by some polynomial n^k, then we choose k_2 such that *RANDOM* works with probability $1 - O(1/n^k)$.

5. UNIVERSALITY

The performance of the routing algorithm *RANDOM* allows us to generalize the universality theorem from [Le] that states that a universal fat-tree of a given volume can simulate any other routing network of equal volume with only a polylogarithmic factor increase in the time required. The original proof assumed the simulation of the routing network was off-line. In this section we show that the simulation can be carried out in the more interesting on-line context. We first discuss VLSI models briefly and state how channel capacities can be chosen for area and volume-universal fat-trees. We then give a simple universality result that requires no routing algorithm. Finally, we give a stronger universality theorem based on *RANDOM*.

VLSI models provide a means of measuring hardware costs quantitatively in terms of area or volume [L, LR, Le1, Le2, T].

These models are interesting from an engineering point of view because "pin-boundedness" is modeled directly as the limitation on communication imposed by the perimeter of a two-dimensional region or by the surface area of a three-dimensional region. In VLSI models, the processors and wires of a network are the vertices and edges of a graph. The graph must be embedded in a two- or three-dimensional grid such that vertices are mapped to gridpoints and edges are mapped to disjoint paths in the grid. The area or volume of the network is the number of gridpoints occupied by either vertices or edge segments. These assumptions implicitly restrict the number of connections to a processor to at most four in two dimensions and six in three dimensions, but generalizations to larger processors are straightforward.

In order for a fat-tree to be universal for area or volume, the channel capacities must be picked properly. Let us consider area, instead of volume, for simplicity. Intuitively, we wish the processors to be densely packed in the region required by the network. The bandwidth of communication to a region is constrained by the perimeter of the region, however, and thus if the channel capacity to a subtree is too large relative to the number of processors in the subtree, the processors will not be densely packed. On the other hand, if we choose the channel capacities too small, the processors will indeed be densely packed, but we will not take maximal advantage of the available communication bandwidth. Consequently, we choose the capacity of a given channel to be proportional to the perimeter of the square region the processors rooted at that channel would occupy if there were no wires, that is $\Theta(\sqrt{n})$ if the subtree has n processors. It turns out that the additional area required by the wires in the channels does not greatly increase the area beyond that required by the processors alone: the area for n processors is $\Theta(n \lg^2 n)$, the same as that required by Leighton's tree of meshes graph [L].

Following this intuition, an area-universal fat-tree can be constructed by giving each leaf channel a constant capacity, and then growing the channel capacities by $\sqrt{2}$ at each level as we go up the tree, rounding off to integer capacities. Another scheme that avoids rounding is to double the channel capacities every two levels, as is done in the fat-tree of Figure 1. Either of these methods yields a $\Theta(n \lg^2 n)$-area layout for n processors, and a root capacity of $\Theta(\sqrt{n})$. Volume-universal fat-trees can be constructed in a similar fashion by picking a growth rate of $\sqrt[3]{4}$, or equivalently, by quadrupling the

capacity every three levels. The volume of an n-processor fat-tree constructed by these methods is $\Theta(n \lg^{3/2} n)$, and the root capacity is $\Theta(n^{2/3})$, as can be shown with the arguments in [LR] or [Le2].

Even without a good routing algorithm for fat-trees, it is possible to prove a simple universality property. The theorem is presented for area-universal fat-trees—a similar theorem holds for volume-universal ones.

THEOREM 8. *Let R be a interconnection network occupying a square of area n such that all connections are point-to-point between processors with no intervening switches. Then an area-universal fat-tree of area $O(n \lg^2 n)$ can simulate every step of network R with at most $O(\lg^2 n)$ switching delay.*

Proof. We use an area-universal fat-tree such as that shown in Figure 1, where the channel capacity to leaves is 4 and the root capacity is $4\sqrt{n}$. Network R lies in a square with side length $\sqrt{n}$. Each processor of R is mapped to the corresponding processor of the fat-tree in the natural geometric fashion. This mapping satisfies the property that the capacity of any channel of the fat-tree is at least as great as the perimeter of the corresponding region of the layout of network R. Therefore, any communication step performed by R induces at most a load factor of 1 on the fat-tree and thus can be routed in one delivery cycle. Since each delivery cycle requires only $O(\lg^2 n)$ bit times, the theorem follows. □

This universality result is weak in several ways. For example, the fat-tree network occupies slightly more area than the simulated network R. It seems reasonable to compare networks of exactly equal cost. Another weakness in the result is that it forbids networks with intermediate switches that buffer messages for several time steps. We could model switches as processors, but in some contexts, processors might be considerably more expensive than switches. Since we can directly route messages sets with large load factors using *RANDOM*, we can prove a stronger universality result that addresses these concerns. The next theorem, a generalization of that in [Le2], is presented for volume-universal fat-trees. One can prove an analogous theorem for area-universal fat-trees.

THEOREM 9. *Let FT be a volume-universal fat-tree of volume v, and let R be an arbitrary routing network also of volume v on a set*

of $n = O(v/\lg^{3/2} v)$ *processors. Then the processors of R can be mapped to processors of FT such that any message set M that can be delivered in time t by R can be delivered by FT in time* $O((t + \lg\lg n)\lg^3 n)$ *with probability* $1 - O(1/n)$.

Proof. The proof parallels that of [Le2]. The reader is referred to that paper for details. The routing network R of volume v is mapped to FT in such a way that any message set M that can be delivered in time t by R puts a load factor of at most $O(t \lg(n/v^{2/3}))$ on FT. By Theorem 7, the message set M can be delivered by $RANDOM$ in $O(t \lg(n/v^{2/3}) + \lg n \lg\lg n)$ delivery cycles with high probability. Since each cycle takes at most $O(\lg^2 n)$ time, the result follows. □

6. GREEDY STRATEGIES

It is natural to wonder whether a simple greedy strategy of sending all undelivered messages on each delivery cycle, and letting them battle their way through the switches, might be as effective as $RANDOM$, which we have shown to work well on every message set. As a practical matter, a greedy strategy may be a good choice, but it seems difficult to obtain tight bounds on the running time of greedy strategies. In fact, we show in this section that no naive greedy strategy works as well as $RANDOM$ in terms of asymptotic running times. For simplicity, we restrict our proof to deterministic strategies and comment later on the extension to probabilistic ones. Specifically, we show that for a wide class of deterministic greedy strategies, there exist n-processor fat-trees and message sets with load factor λ such that $\Omega(\lambda \lg n)$ delivery cycles are required. Thus, if λ is asymptotically larger than $\lg\lg n$, the greedy strategy is worse than $RANDOM$, which essentially guarantees $O(\lambda + \lg n \lg\lg n)$ delivery cycles for any set of messages. The lower bound proof for greedy routing is based on an idea due to F. M. Maley [M].

Figure 5 shows the greedy algorithm. The code for $GREEDY$ does not completely specify the behavior of message routing on a fat-tree because the switches have a choice as to which messages to drop when there is congestion. (The processors also have this choice, but we shall think of them as being switches as well.) In the analysis of $RANDOM$, we presumed that all messages in the channel are lost if the channel is congested. To completely specify

Figure 5. The algorithm *GREEDY* for delivering a message set M. This algorithm repeatedly sends all undelivered messages. The performance is highly dependent on the behavior of the switches.

```
1   while M ≠ ∅ do
2       send M
3       M ← M − {messages delivered}
4   endwhile
```

the behavior of *GREEDY*, we must define the behavior of switches when channels are congested.

The lower bound for *GREEDY* covers a wide range of switch behaviors. Specifically, we assume the switches have the following two properties:

1. Each switch is greedy in that it drops messages only if a channel is congested, and then only the minimum number necessary.
2. Each switch is *oblivious* in that decisions on which messages to drop are not based on any knowledge of the message set other than the presence or absence of messages on the switch's input lines.

We define the switches of a fat-tree to be *admissible* if they have these two properties. The conditions are satisfied, for example, by switches that drop excess messages at random, or by switches that favor one input channel over another. An admissible switch can even base its decisions on previous decisions, but it cannot predict the future or make decisions based on knowing what (or how many) messages it or other switches have dropped. (The definition of oblivious in property 2 can be weakened to include an even wider range of switch behaviors without substantially affecting our results.)

At this point, we restrict attention to deterministic greedy strategies and present the lower bound theorem for *GREEDY* operating on an area-universal fat-tree. The theorem can be extended to a variety of other fat-trees. A discussion of the extension to probabilistic greedy strategies follows the proof of the theorem.

THEOREM 10. *Consider an n-processor area-universal fat-tree with deterministic admissible switches whose channel capacities are* 1

nearest the processors and double at every other level going up to the tree. Then there exist message sets with load factor λ for which GREEDY requires $\Omega(\lambda \lg n)$ delivery cycles.

Proof. For any $\lambda \geqslant 12$, we will construct a "bad" message set M_n on the n processors of the fat-tree by induction on the subtrees. The message set M_n will consist entirely of messages to be routed out of the root and will satisfy the following three properties:

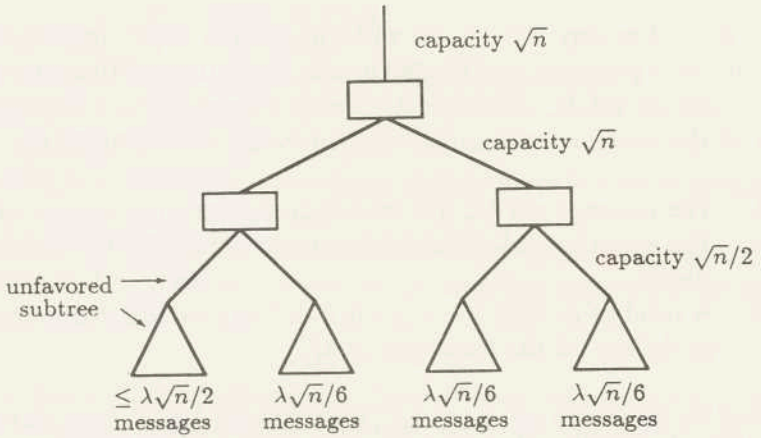
1. The message set M_n has load factor at most λ.
2. The root channel of the fat-tree is full for the first $\frac{1}{3}\lambda$ delivery cycles.
3. A total of at least $\frac{1}{6}\lambda + \frac{1}{12}\lambda \lg n$ delivery cycles[3] are required to deliver all the messages in M_n.

For the base case we consider a subtree with 1 processor, that is, a leaf connected to a channel of capacity 1. The bad message set M_1 consists of λ messages to be sent from the single processor. The properties are satisfied since the root channel is congested throughout the first $\frac{1}{3}\lambda$ delivery cycles, and at least $\frac{1}{6}\lambda$ delivery cycles are needed to deliver all the messages.

We next show that we can construct the bad message set M_n assuming that we can construct a bad message set for a subtree of $n/4$ processors. The construction uses an adversary argument. First, we specify the pattern of inputs seen by the root switch of the fat-tree during certain delivery cycles, without giving any indication of how that input pattern can be achieved. Then since we have given enough information to determine the behavior of the root switch during these cycles, the root switch must announce which messages it passes through to its output. Finally, we give a construction for a message set that achieves the input pattern we called for in the first step. We take advantage of the announced behavior of the root switch in order to ensure that the message set also satisfies properties 1, 2, and 3.

We begin by calling for the input channels of the root switch of the fat-tree to be full for t delivery cycles, where t is $\frac{1}{3}\lambda$. If this is achieved, the total number of messages removed from the fat-tree during the first t delivery cycles is $m = \frac{1}{3}\lambda\sqrt{n}$, since the root capacity is $\sqrt{n}$ and the root switch is greedy. Also, as mentioned before, the specified input pattern determines the behavior of the root switch because the switch is oblivious.

Figure 6. Construction of M_n for the proof of Theorem 10. The subtree from which the fewest number of messages have been delivered by a certain time is loaded with the largest number of messages.



The behavior of the root switch determines how many of the m messages removed from the fat-tree by delivery cycle t come from each of the four subtrees shown in Figure 6. At least one of these subtrees provides no more than $m/4$ of the messages. We choose one such subtree and refer to it as the *unfavored* subtree. The other subtrees are referred to as the *favored* subtrees.

Having determined the unfavored subtree given the conditions specified so far, we can complete the construction of M_n. The unfavored subtree contains a copy of the bad message set $M_{n/4}$ for that subtree. Each of the other three subtrees contains $\frac{1}{6}\lambda\sqrt{n}$ messages evenly divided among the processors in the subtree. Now we must prove that M_n meets all of our requirements.

First, we show that M_n is consistent with the input pattern specified for the root switch. To show that the input channels of the root switch of the fat-tree are full through the first t delivery cycles, it suffices to prove that the root channels of the four subtrees are full through this time. The root channel of the unfavored subtree is full by the induction hypothesis (property 2). The root channel of each favored subtree is also full for the first t delivery cycles, since its messages are evenly distributed, its switches are greedy, and t times its root capacity does not exceed the number of messages emanating from it.

We now prove that properties 1, 2, and 3 hold for M_n. The load factor in the favored subtree is less than λ by construction.

The load factor is at most λ in the unfavored subtree by the induction hypothesis (property 1), so the number of messages in the unfavored subtree is at most $\frac{1}{2}\lambda\sqrt{n}$, and the total number of messages in M_n is at most

$$\tfrac{1}{2}\lambda\sqrt{n} + 3 \cdot \tfrac{1}{6}\lambda\sqrt{n} = \lambda\sqrt{n}.$$

Thus, the load factor of M_n on the fat-tree is at most λ, and property 1 holds. Property 2 is satisfied for M_n because the root switch is greedy. We have already shown that the input channels of the root switch are full through delivery cycle t, so the root channel is certainly full for the required amount of time. Finally, property 3 holds because after running $t = \frac{1}{3}\lambda$ delivery cycles, only $m/4 = \frac{1}{12}\lambda\sqrt{n}$ messages have been removed from the unfavored subtree. If priority had been given to the unfavored subtree, only $\frac{1}{6}\lambda$ delivery cycles would have been required to remove the $m/4$ messages. So by the induction hypothesis (property 3), an additional $\frac{1}{6}\lambda + \frac{1}{12}\lambda \lg(n/4) - \frac{1}{6}\lambda$ cycles are required to empty the unfavored subtree. If we include the original t cycles, the total number of cycles required to deliver all the messages in M_n is at least $\frac{1}{6}\lambda + \frac{1}{12}\lambda \lg n$. □

When probabilistic admissible switches are permitted, the proof of Theorem 10 can be extended to show that the expected number of delivery cycles is $\Omega(\lambda \lg n)$. The idea is that at least one of the subtrees in Figure 6 must be unfavored with probability at least 1/4. We call one such subtree the *often-unfavored* subtree. The construction of M_n proceeds as before, with the often-unfavored subtrees playing the previous role of the unfavored subtrees. In any particular run of *GREEDY*, we expect 1/4 of the often-unfavored subtrees to be unfavored, so there is a $\Theta(1)$ probability that 1/8 of the often-unfavored subtrees are unfavored (Lemma 3). Thus, the probability is $\Theta(1)$ that $\Omega(\lambda \lg n)$ delivery cycles are required, which means that the expected number of delivery cycles is $\Omega(\lambda \lg n)$.

Although we have shown an unfavorable comparison of *GREEDY* to *RANDOM*, it should be noted that *GREEDY* does achieve the lower bound we proved for routing messages out the root. That is, routing of messages out the root or, more generally, up the tree only, can be accomplished by *GREEDY* in $O(\lambda \lg n)$ delivery cycles. This can be seen by observing that the highest congested channel (closest to the root) must drop at least one level every λ delivery

cycles. If one could establish an upper bound of λ times a polylogarithmic factor for the overall problem of greedy routing, it would show that *GREEDY* still has merit despite its inferior performance in comparison to *RANDOM*.

7. FURTHER RESULTS

This section contains three additional results relevant to routing on fat-trees. The first is a simple randomized algorithm for routing on fat-trees in which each channel has at least logarithmic capacity. The second is a new class of fat-trees that have much simpler switches than the ones thus far considered. The final result is an improvement to the off-line routing result of [Le2].

7.1. Larger Channel Capacities

We can improve the results for on-line routing if each channel c in the fat-tree is sufficiently large, that is if $\text{cap}(c) = \Omega(\lg n)$. Specifically, we can deliver a message set M in $O(\lambda(M))$ delivery cycles with high probability, i.e., we can meet the lower bound to within a constant factor. The better bound is achieved by the algorithm *RANDOM'* shown in Figure 7.

THEOREM 11. *For any message set M on an n-processor fat-tree with channels of capacity $\Omega(\lg n)$, the probability is at least $1 - O(1/n)$ that RANDOM' will deliver all the messages of M in $O(\lambda(M))$ delivery cycles, if $\lambda(M)$ is polynomially bounded.*

Figure 7. The algorithm *RANDOM'* for routing in a fat-tree with channels of capacity $\Omega(\lg n)$. This algorithm repeatedly doubles a guessed number of delivery cycles, z. For each guess, each message is randomly sent in one of the delivery cycles.

```
1   z ← 1
2   while M ≠ ∅ do
3       for each message m ∈ M, choose a random number i_m ∈ {1, 2, ..., z}
4       for i ← 1 to z do
5           send all messages m such that i_m = i
6       endfor
7       z ← 2z
8   endwhile
```

Proof. Let the lower bound on channel size be $a \lg n$, and let n^k be the polynomial bound on the load factor $\lambda(M)$. We consider only the pass of the algorithm when z first exceeds $e2^{(k+2)/a}\lambda(M)$. We ignore previous cycles for the analysis of message routing, except to note that the number of delivery cycles they require is $O(\lambda(M))$.

We first consider a single channel c within a single cycle i from among the z delivery cycles in the pass. Since each message has probability $1/z$ of being sent in cycle i, we can apply Lemma 4 with $p = 1/z$ to conclude that the probability that channel c is congested in cycle i is at most

$$\begin{aligned}\left(\frac{e\lambda(M)}{z}\right)^{\mathrm{cap}(c)} &\leqslant 2^{-\lfloor (k+2)/a \rfloor \mathrm{cap}(c)} \\ &\leqslant 2^{-(k+2)\lg n} \\ &= \frac{1}{n^{k+2}}.\end{aligned}$$

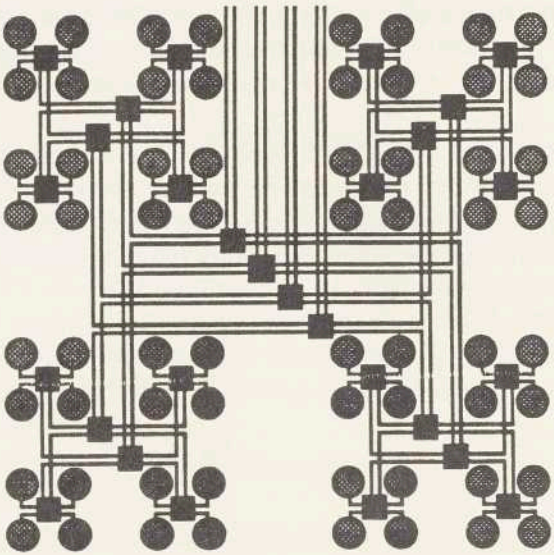
Since there are $O(n)$ channels, the probability that there exists a congested channel in cycle i is $O(1/n^{k+1})$. Finally, since there are $z \leqslant 2e2^{(k+2)/a}\lambda(M) = O(\lambda(M)) = O(n^k)$ cycles, the probability is $O(1/n)$ that there exists a congested channel in any delivery cycle of the pass. □

7.2. Another Universal Fat-Tree

We have recently discovered a fat-tree design that uses simpler switches than the fat-tree described in Section 1 and in [Le2]. Figure 8 illustrates the structure of a two-dimensional universal fat-tree of this new type. Each of the switches in this fat-tree can switch messages among four child switches and two parent switches. The area of the fat-tree is $\Theta(n \lg^2 n)$.[4] In three dimensions, we can use switches with eight children and four parents to obtain a fat-tree with volume $\Theta(n \lg^{3/2} n)$.

The new fat-tree design satisfies the universality property of Theorem 9, except that the degradation in time is $O(\lg^4 n)$. The new fat-tree structure removes a factor of $\lg n$ from the time to perform a delivery cycle since the switches have constant depth. The number of delivery cycles needed to route a set M of messages is

Figure 8. Another fat-tree design. The switches in this structure have constant size.



$O(\lambda(M)\lg^2 n)$, however, which yields $O(\lambda(M)\lg^3 n)$ total time, as compared with $O((\lambda(M) + \lg n \lg\lg n)\lg^2 n)$ for the original fat-tree.

The mechanics of routing on the new fat-tree are somewhat different than on the original. The underlying channel structure for the two fat-trees is the same, but the new fat-tree does not rely on concentrators to make efficient use of the available output wires. Instead, each message sent through the fat-tree randomly chooses which parent to go to next (based on random bits embedded in its address field) until it reaches the apex of its path, and then it takes the unique path downward to its destination. This strategy guarantees that for any given channel through which a message must pass, the message has an equal likelihood of picking any wire in the channel.

The routing algorithm is a modification of the algorithm *RANDOM′*. We simply surround lines 3–6 with a loop that executes these lines $(k + 1)\lg n$ times, where $|M| = O(n^k)$.

The proof that the algorithm works applies the analysis from Section 4 to individual wires, treating them as channels of capacity 1. Consider a wire w traversed by a message in a p-subset M' of M, and consider the channel c that contains the wire. For any other message in M, the probability is $p/\text{cap}(c)$ that the message is directed

to wire w when the message set M' is sent. Thus, the probability that w is congested is at most $B(1, \text{load}(M, c), p/\text{cap}(c)) \leqslant ep\lambda(M)$, and an analogue to Lemma 4 holds because the capacity of w is 1. Lemma 5, which says that the probability is $\frac{1}{2}p$ that a given message of M is delivered when a p-subset of M is sent, also holds if the congestion parameter r is chosen to be $\Theta(\lg n)$.

We can now prove a bound of $O(\lambda(M)\lg^2 n)$ on the number of delivery cycles required by the algorithm to deliver all the messages in M. It suffices to show that with high probability, all the messages in M get routed when the variable z in the algorithm reaches $\Theta(\lambda(M)\lg n)$. When $z \geqslant r\lambda(M) = \Theta(\lambda(M)\lg n)$, any given message m is sent once during a single pass through lines 3–6, and the probability that the message is not delivered on that pass is at most $\frac{1}{2}$. Thus, the probability that m is not delivered on any of the $(k+1)\lg n$ passes through lines 3–6 is at most $1/n^{k+1}$. Since the number of messages in M is $O(n^k)$, the probability is $O(1/n)$ that a message exists that is not routed by the time z reaches $r\lambda(M)$.

7.3. Off-Line Routing

Our analysis for *RANDOM* has ramifications for the off-line routing problem. We have shown that with high probability, the number of delivery cycles given by Figure 4 suffices to deliver a message set with load factor λ. Consequently, there must exist off-line schedules using only this many delivery cycles, which improves the bound of $O(\lambda \lg n)$ given in [Le2]. The previous off-line bound was proved by giving a deterministic, polynomial-time construction of a routing schedule that achieves the bound. Whether a deterministic, polynomial-time algorithm exists that achieves our better bound is an open question.

Perhaps the bound on off-line routing can be further improved (e.g., to $O(\lambda + \lg n)$). The integer programming framework of Raghavan and Thompson [RT] is one possible approach that might give a probabilistic construction that achieves this bound. On the other hand, it may be possible to apply more direct combinatorial techniques to yield an improved deterministic bound.

8. CONCLUDING REMARKS

This chapter has studied the problem of routing messages on fat-tree networks. We have obtained good bounds for randomized

routing based on the load factor of a set of messages. Our algorithms directly address the problem of message congestion and require no intermediate buffering, unlike many algorithms in the literature. We have shown how to use the routing algorithms to prove that fat-trees are volume-universal networks. This section discusses some directions for future research.

The analysis of the algorithm *RANDOM* gives reasonably tight asymptotic bounds on its performance, but the constant factors in the analysis are large. In practice, smmaller constants probably suffice, but it is difficult to simulate the algorithm to determine what constants might be better. Unlike Valiant's algorithm for routing on the hypercube, our algorithm does not have the same probabilistic behavior on all sets of messages, and, therefore, the simulation results may be highly correlated with the specific message sets chosen. The search for good constants is thus a multidimensional search in a large space, where each data point represents an expensive simulation.

Although we have shown that *GREEDY* is asymptotically worse than *RANDOM*, it may be that it is more practical to implement. The logarithmic-factor overhead that we have been able to show is mitigated by a constant factor of $\frac{1}{12}$. Simulations indicate that a greedy algorithm might actually work quite well [I], but we have been unable to prove a good upper bound on its performance. Despite the simplicity of control offered by *GREEDY*, it seems unwise to base the design of a large, parallel supercomputer on unproven conjectures of performance. Thus, a comprehensive analysis of *GREEDY* remains an important open problem.

The idea of using load factors to analyze arbitrary networks is a natural one. We have been successful in analyzing fat-trees using this measure of routing difficulty. It may be possible to analyze other networks in terms of load factor, but some improvement to our techniques seems to be necessary if channel widths are small and the diameter of the network is large. The problem is that a message that passes through many small channels has a high likelihood of conflicting with other messages. One solution might involve buffering messages in intermediate processors or switches.

The high probability results reported in this paper for routing on fat-trees are almost deterministic in the sense that substantial deviation from the expected performance will probably never occur in one's lifetime. On the other hand, from a theoretical point of view, it would be nice to match the results of this paper with truly

deterministic algorithms. Most deterministic routing algorithms in the literature are based on sorting, and thus a direct application to fat-trees causes congestion problems, much as does Valiant's routing technique. A deterministic routing algorithm for fat-trees that circumvents these problems would yield even stronger universality properties than we have shown here.

ACKNOWLEDGMENTS

We have benefited tremendously from the helpful discussions and technical assistance of members of the theory of computation group at MIT. Thanks to Ravi Boppana, Thang Bui, Benny Chor, Peter Elias, Oded Goldreich, Johan Hastad, Alex Ishii, Tom Leighton, Bruce Maggs, Miller Maley, Cindy Phillips, Ron Rivest, and Peter Shor.

This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-80-C-0622 and in part by the Office of Naval Research under Contract N00014-86-K-0593. Ron Greenberg was supported in part by a Fannie and John Hertz Foundation Fellowship. Charles Leiserson is supported in part by an NSF Presidential Young Investigator Award.

NOTES

1. In this chapter, we measure time in terms of bit operations, rather than word operations, to better reflect actual costs.

2. We use the weak bound of Lemma 3 because we cannot assume that the probability that a message is delivered in a given cycle is independent of the probabilities for other messages. In practice, one would anticipate that the dependencies between messages are weak, and that the algorithm would be effective with much smaller values for the constants k_1 and k_2 than we prove here.

3. Without loss of generality, we assume henceforth that $\frac{1}{12}\lambda$ is integral, since we could otherwise use $\lfloor \frac{1}{12}\lambda \rfloor$ with only a constant factor change.

4. Interestingly, a mesh-of-trees [L] can be directly embedded in this fat-tree. In fact, it can be shown using sorting arguments that a mesh-of-trees is area-universal [LL].

REFERENCES

[Aj] M. Ajtai, J. Komlós, and E. Szemerédi, "Sorting in $c \log n$ parallel steps," *Combinatorica* 3(1): 1–19 (1983).

[Al] R. Aleliunas, "Randomized parallel communication," *Proc. 1st Annu. ACM Symp. Principles Distributed Computing* 60–72 (1982).

[B] V. E. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, New York, 1965.

[C] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Ann. Math. Statistics* 23: 493–507 (1952).

[F] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, 2nd ed. John Wiley, New York, 1957.

[I] A. T. Ishii, *Interprocessor Communication Issues in Fat-Tree Architectures*, Bachelor's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1985.

[LF] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *JACM* 27(4): 831–838 (1980).

[L] F. T. Leighton, *Complexity Issues in VLSI*. MIT Press, Cambridge, 1983.

[LL] F.T. Leighton and C. E. Leiserson, private communication, May 1985.

[LR] F. T. Leighton and A. L. Rosenberg, "Three-dimensional circuit layouts," *SIAM J. Computing* 15(3): 793–813 (1986).

[Le1] C. E. Leiserson, *Area-Efficient VLSI Computation*. MIT Press, Cambridge, 1983.

[Le2] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transact. Computers* C-34(10): 892–901 (1985).

[M] F. M. Maley, private communication, October 1984.

[P] N. Pippenger, "Parallel communication with limited buffers," *Proc. 25th Annu. Symp. Foundations Computer Sci. IEEE* 127–136 (1984).

[RT] P. Raghavan and C. D. Thompson, "Provably good routing in graphs: Regular arrays," *Proc. 17th Annu. ACM Symp. Theory Computing* 79–87 (1985).

[T] C. D. Thompson, *A Complexity Theory for VLSI*, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[U] E. Upfal, "Efficient schemes for parallel communication," *JACM* 31(3): 507–517 (1984).

[V] L. G. Valiant, "A scheme for fast parallel communication," *SIAM J. Computing* 11(2): 350–361 (1982).

[VB] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," *Proc. 13th Annu. ACM Symp. Theory Computing* 263–277 (May 1981).

FACTORIZATION OF POLYNOMIALS GIVEN BY STRAIGHT-LINE PROGRAMS

Erich Kaltofen

ABSTRACT

An algorithm is developed for the factorization of a multivariate polynomial represented by a straight-line program into its irreducible factors. The algorithm is in random polynomial-time as a function in the input size, total degree, and binary coefficient length for the usual coefficient fields and outputs a straight-line program, which with controllably high probability correctly determines the irreducible factors. It also returns the probably correct multiplicities of each distinct factor. If the coefficient field has finite characteristic p and p divides the multiplicities of some irreducible factors our algorithm constructs straight-line programs for the appropriate pth powers of such factors.

Also a probabilistic algorithm is presented that allows a polynomial given by a straight line program to be converted into its sparse

Advances in Computing Research, Volume 5, pages 375–412.

ISBN: 0-89232-896-7

representation. This conversion algorithm is in random-polynomial time in the previously cited parameters and in an upper bound for the number of nonzero monomials permitted in the sparse output. Together with our factorization algorithm we therefore can probabilistically determine all those sparse irreducible factors of a polynomial given by a straight-line program that have less than a given number of monomials. We show that this result is valid without any restriction to the characteristic of the coefficient field.

The first section of this chapter also summarizes the history of the polynomial factorization problem, and the last section discusses what questions for this problem remain to be solved. We have also attempted to provide an extensive list of references on the subject, so that this chapter can serve as a starting point for someone without previous knowledge in polynomial factorization.

1. THE PROBLEM OF FACTORING POLYNOMIALS

"The invention of divisors of universal quantities," what we refer to today as the computation of factors of polynomials, was already taught by Newton in 1673 and the method was subsequently published in his *Arithmetica Universalis* [N]. In 1882 Kronecker [Kr, pp. 10–13] reduced the problem of factoring multivariate polynomials over algebraic number fields to factoring univariate polynomials over the integers, for which he applied Newton's algorithm. van der Waerden's influential text [vW] discusses those algorithms and suggests that for larger problems they are not very practical. Nonetheless, early computer programs realized this classical approach [JK] and verified that it is quite inefficient. The ensuring search for efficient algorithms to factor polynomials is a fine example in the discipline of the design and analysis of algorithms as well as complexity theory and exhibits many of the techniques developed for these subjects.

In 1967 Berlekamp [B1] found an algorithm to factor univariate polynomials over moderately sized finite fields in time proportional to the cube of the input degrees. Berlekamp's algorithm is the first evidence that polynomial factorization is not as complex a problem as is integer factoring. However, his algorithm performed badly when applied to large finite fields. Berlekamp's own resolution of this problem in 1970 [B2] is remarkable in that by introducing the selection of random field elements the algorithm could be exponentially sped up. Thus the factorization algorithm over large

finite fields became one of the forerunners of "randomized" algorithms. We also refer to Rabin's 1976 version of this algorithm [R2] for his appealing probability analysis, and to the book [Kn, §4.6.2] for a discussion of additional work. Recently, the problem of removing the random choices from the algorithm without sacrificing polynomial running time has been resolved for several special cases by the use of interesting new ideas, and we refer to the two exemplary papers [Sh] and [Hu]. The performance in practice of the randomized algorithms for univariate polynomial factorization over large finite fields is quite satisfactory and, at the moment, far superior to any known deterministic algorithms.

The advances in factoring polynomials modulo a prime integer suggested to apply these algorithms to factoring polynomials with integer coefficients as well. Zassenhaus in 1969 [Z] pointed to the "Hensel Lemma" [Hen, §4] as a means to reconstruct the integral factors from modular ones. Unlike the factorization algorithm for polynomials over finite fields, however, the reconstruction procedure for the integral polynomial factors from the modular ones can have exponential complexity due to "combinatorical explosion" [B2, K11]. "Probabilistic analysis" [M2, and Cd] shows, however, that this problem does not arise on "average" inputs and implementations of the Berlekamp–Hensel algorithm for factoring univariate integral polynomials perform quite well, except for very special inputs. However, such inputs can arise and are, in fact, generated by Kronecker's reduction from algebraic number fields, for instance. In 1982 a remarkable diophantine algorithm was found by Lenstra, Lenstra, and Lovász [LLL] to overcome the combinatorical explosion by a polynomial-time construction. Several more classical problems could then be shown to also belong to the polynomial-time complexity class, for example, solvability by radicals [LM], factorization of univariate polynomials over algebraic number fields [LT, L], and the multivariate polynomial factorization.

Already in 1971 Musser [M1] demonstrated that "Hensel Lifting," as the procedure applying the Hensel lemma is now called, can be also applied to reconstruct multivariate from univariate factors. Combinatorical explosion is still a problem, but not all mappings to the univariate factorization are categorically bad. This is a consequence of the famous Hilbert Irreducibility Theorem [Hi], and the first polynomial-time reduction by Kaltofen found in late 1981 is based on an effective deterministic version of that theorem [K1, also in K3, §7]. A number of different polynomial-time reductions from

multivariate to univariate polynomial factorization are known today [CG, GK1, K3, K9, Le2, Le3]. All these algorithms assume that all possible terms count toward the input size, in other words the multivariate polynomials are represented "densely."

If the number of variables in the multivariate factorization problem is allowed to grow with the problem size, then the "sparseness" of the input and output polynomials need to be take into consideration. Wang, upon considering the very sparse examples presented by Claybrook [Cl], invented several heuristics to cope with the intermediate "expression swell" occurring for sparse inputs and outputs [W]. Zippel in 1979 carried these considerations further by introducing randomization into the Hensel lifting process [Zi2]. In order to make a rigorous analysis of the failure probabilities, an effective probabilistic version of the Hilbert irreducibility theorem was needed. Although Heintz and Sieveking [HS] had already provided such a theorem for algebraically closed fields, in 1983 von zur Gathen provided a suitable version for arbitrary coefficient fields [G2] and applied it to the sparse factoring problem [GK2]. In retrospect, Kaltofen's effective Hilbert irreducibility theorem also lent itself to an even simpler probabilistic version [K4]. In [GK2] sparse polynomials are described that possess irreducible factors with superpolynomially more terms. These examples imply that any sparse Hensel Lifting scheme can have more than polynomial running time on certain inputs. It became clear that to deal with this phenomenon the sparse representation had to be replaced by a more powerful one.

The usage of "straight-line programs" as a means to compute certain polynomials has been developed in the framework of complexity theory in the past decade; refer for example to [S2, S3, PS, S4, Sc, He]. In 1983 von zur Gathan [G2] combined his probabilistic Hilbert irreducibility theorem with the probabilistic method of straight-line program evaluation [S2, IM] to find the factor degree pattern of polynomials defined by straight-line computations. A previously known operation on polynomials in straight-line representation is that of taking first-order partial derivatives [BS]. Although there is evidence that other operations such as higher partial derivatives are inherently complex [V2], the greatest common divisor problem of polynomials in straight-line program representation is in probabilistic polynomial-time, as shown by Kaltofen in 1985 [K7]. In this chapter we show that straight-line programs for the irreducible factors of a polynomial given by a straight-line program can also be found in probabilistic polynomial

straight-line program can also be found in probabilistic polynomial time. With Zippel's 1979 sparse polynomial interpolation algorithm [Zi1] our factorization result resolves all problems left open in [Zi2, GK2, K5]. We note that, unlike the randomized solutions for factorization of univariate polynomials over large finite fields, the probabilistic solutions are of the Monte Carlo kind, "probably correct and always fast." The failure probability can, of course, be made arbitrarily small.

2. DISCUSSION OF RESULTS

A straight-line program is a sequence of arithmetic assignments to new variables, the operands of which are either constants, indeterminates, or previously assigned variables. The operators allowed are addition, subtraction, multiplication, and division. Our algorithms treat this straight-line program as a data structure to represent the polynomials computed by them. This representation can define in polynomial-space families of polynomials with exponentially many individual terms, such as determinants by Gaussian elimination sequences. Unlike algebraic complexity theory applications, in which a straight-line program is a model of computation, our algorithms must not only produce straight-line results of polynomial-length but also perform the transformations efficiently, that is, in random polynomial-time.

It appears proper that we explicitly define the model of algebraic computation in which our algorithms can be formulated. Our model is the *sequential probabilistic algebraic random access machine* (RAM), with which we not only manage computations over an abstract algebraic domain but also resolve the question of random element selections from the abstract fields. For concrete domains such as the rational numbers we also establish binary polynomial running time, even if the algorithms would then be formulated on the probabilistic Turing machine model. We think that the algebraic RAM model is in the spirit of new algebraic computing languages such as Scratchpad II [Je].

The factorization algorithm presented here takes as input a straight-line program computing a polynomial and outputs a straight-line program and multiplicities for irreducible polynomials that, with controllably small error probability, determine the irreducible factors of the input polynomial. If the multiplicities are divisible by the characteristic of the coefficient field, our output is slightly different. The factorization algorithm calls a bivariate

polynomial factorization procedure and is therefore effective and of polynomial running time only for the usual coefficient fields. We measure the running time as a function in the input size and input polynomial degree. Over the rationals, for instance, we get an algorithm of binary complexity that is a polynomial function in the binary size of the straight-line program determining the input polynomial, in its total degree and the size of the numerators and the common denominator of its rational coefficients, and in the logarithm of the inverse of the probability bound that the output program incorrectly determines the irreducible factors or their multiplicities.

The key idea of our algorithm, which we will present and analyze in Section 5, in addition to previously known approaches, is to employ Hensel lifting but to replace the p-adic expansion of the coefficients by the expansions into homogeneous parts of the minor variables. We thus lift all minor variables simultaneously and avoid the variable by variable lifting loop that compounds programs of exponential size. This method can be viewed as a combination of Strassen's trick for eliminating divisions in straight-line computations [S2] and Yun's Hensel lifting scheme [Y]. If the coefficient field is of positive characteristic p and the multiplicity of an irreducible factor is divisible by p, an additional problem arises. We can, however, compute a straight-line computation for the appropriate p^kth power of such a factor.

For completeness in Section 3 we present our version of Zippel's conversion algorithm [Zi1] from straight-line to sparse polynomial representation. Our algorithm is of polynomial complexity in the size of the straight-line program defining the input polynomial, in its total degree, and in an upper bound t for the maximal number of monomials permitted in the sparse output. The algorithm produces either a sparsely represented polynomial with no more than t monomials or a message indicating that the input polynomial has more than t terms. The algorithm is Monte Carlo and can give a wrong answer, with controllably small probability. Over the rational numbers the algorithm is also of binary polynomial running time in the coefficient size of the input polynomial and the logarithm of the inverse of the failure probability. We believe that our conversion algorithm is a general and useful way in which Zippel's sparse interpolation scheme can be formulated.

Let us now come back to the question of factorizing into sparse polynomials. The examples causing superpolynomial blow-up for the size of the answer have the property that many other factors

are very sparse. In general, one may wish to retrieve the sparse factors as such and leave the dense factors in straight-line format. Fortunately, the sparse conversion algorithm discussed allows us to do just that. More precisely, given a bound t we now can probabilistically determine in polynomial time also in t the sparse format of all irreducible factors with no more than t terms, without any restriction on characteristic and multiplicities. Moreover, the running time is always polynomial even if we were unlucky in our choice of evaluation points. We think that this finally settles the question of sparse factorization in a very satisfactory manner.

The next section introduces the model of probabilistic algebraic RAMs, defines straight-line programs, and summarizes results needed from other sources. In Section 4 we present the conversion algorithm to sparse format and in Section 5 the theorems on probabilistically preserving the factor degree pattern. Section 6 contains the straight-line polynomial factorization algorithm. We conclude in Section 7 with a discussion of open problems in connection with the polynomial factorization problem.

3. DEFINITIONS AND PREVIOUS RESULTS

We now repeat the main notions and results presented in [K7]. We denote the field of rational numbers by $\mathbf{Q}$ and the finite field with q elements by $\mathbf{F}_q$. An *algebraic RAM over F*, with F a field, has a CPU that is controlled by a finite sequence of labeled instructions and that has access to an infinite address and data memory (see Figure 1).

The split into two memories, one that facilitates pointer manipulation for array processing as well as maintaining a stack for recursive procedures, and another memory in which the algebraic arithmetic is carried out, is also reflected in other models for algebraic computations such as the parallel arithmetic networks in [G3] or the omnipresence of the built-in type Integer in the Scratchpad II language [Je]. Each word in address memory can hold an integral address and each word in data memory can store an element in F. The CPU also has access to an input and an output medium. The instructions in the CPU may have one or two operands that typically are integers. The operands refer to words in address or data memory depending on whether the instruction is an address or a data instruction. Indirect addressing is indicated by a negative operand. For completeness the microcode for a full instruction set is given in Figure 2.

Figure 1. Algebraic RAM over $\mathbf{Q}(\sqrt{2})$.

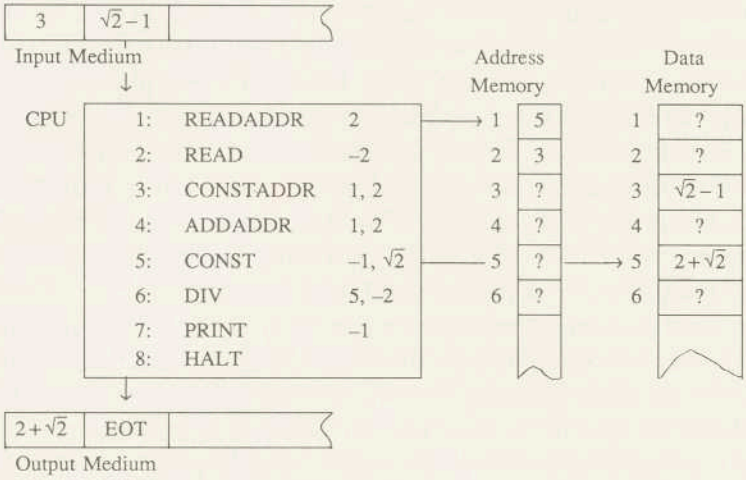


Figure 2. Summary of algebraic RAM instructions.

Instruction		*Description*
ADD[ADDR]	i, j	$Op_i \leftarrow Op_i + Op_j$ (see below).
SUB[ADDR]	i, j	$Op_i \leftarrow Op_i - Op_j$.
MULT[ADDR]	i, j	$Op_i \leftarrow Op_i \times Op_j$.
DIVADDR	i, j	$Op_i \leftarrow \lfloor Op_i / Op_j \rfloor$.
DIV	i, j	$Op_i \leftarrow Op_i / Op_j$.
CONST[ADDR]	i, c	$Op_i \leftarrow c$.
MOVE[ADDR]	i, j	$Op_i \leftarrow Op_j$.
JMP	l	Execution continues at program label l.
JMPZ[ADDR]	i, l	If $Op_i = 0$ then execution continues at program label l.
JMPGZADDR	i, l	If $Op_i > 0$ then execution continues at program label l.
READ[ADDR]	i	The input medium is advanced and the next item is read into Op_i.
PRINT[ADDR]	i	The output medium is advanced and Op_i is written onto the medium.
HALT		An EOT marker is written onto the output tape and execution terminates.

$$Op_i = \begin{cases} \text{AM}[i] \\ \text{DM}[i] \end{cases} \text{ if } i > 0 \text{ and } \begin{cases} \text{address} \\ \text{data} \end{cases} \text{ instruction}$$

$$\begin{cases} \text{AM}[\text{AM}[-i]] \\ \text{DM}[\text{AM}[-i]] \end{cases} \text{ if } i < 0 \text{ and } \begin{cases} \text{address} \\ \text{data} \end{cases} \text{ instruction}$$

AM = address memory, DM = data memory

AM[$-i$] must be positive, otherwise an interrupt occurs.

The *arithmetic time* and *space complexity* of an algebraic RAM for a given input are defined as the number of instructions executed and the highest memory address referenced, respectively. It is not always realistic to charge for each arithmetic operation in F one time unit. We will consider encoding data in binary and define as size(a), $a \in F$, where F is a concrete field such as $\mathbf{Q}$ or $\mathbf{F}_q$, the number of bits needed to represent a. Then the cost and space of an arithmetic instruction depend on the size of its operands. The *binary* time and space complexity of an algebraic RAM over F is derived by charging for each arithmetic step in F as many units as are needed to carry out the computation on a multitape Turing machine. Notice that we generally assume that the field arithmetic can be carried out in polynomial binary complexity with respect to the size of the operands. What that implies in particular is that elements in $\mathbf{F}_q$, say, always require $O(\log(q))$ representation size independent of whether they are residues of small integral values or not. For READ, PRINT, CONST, MOVE, or JMPZ instructions we charge as many units as is the size of the transferred or tested element.

We also apply this "logarithmic cost criterion" to the address computations and assume that every address is represented as a binary integer. The binary cost for performing address arithmetic is again the Turing machine cost. For indirect addressing we add the size of the final address to the binary time and space cost of the corresponding instruction. We note that in most circumstances the binary cost for performing address arithmetic is by far dominated by the binary cost of the algebraic operations and that for all practical purposes the largest storage location is of constant size. But our more precise measure has its advantages. First, all binary polynomial-time algorithms on algebraic RAMS are also polynomial-time in the Turing machine model. Second, the true binary complexity is measured if we can use the address memory for more than address computations, e.g., for hashing with sophisticated signatures. Another such example is that of selecting random field elements.

A *probabilistic* algebraic RAM is endowed with the additional instruction

$$\text{RANDOM}\,\{\text{ADDR}\}\,i, j$$

with the following meaning. Into Op_i an element of F (or address) is stored that was uniformly and randomly polled from a set R of

elements (or integers) with $\mathrm{card}(R)$ equal to the address operand Op_j (see Figure 2 for the definition of Op). The selection of R is unknown except all its elements $a \in R$ have $\mathrm{size}(a) = O(\log Op_j)$. This model of randomized algebraic computation overcomes the problem of how to actually generate a "random" rational number, say, and, as we will show later, the failure probabilities can in our circumstances be fully analyzed. Now we only note that for a nonzero polynomial f the probability

$$\mathrm{Prob}(f(a_1, \ldots, a_n) = 0 \mid a_i \in R) \leqslant \frac{\deg(f)}{\mathrm{card}(R)}, \tag{1}$$

(see [Sw]).

Our algorithms will read as input, produce as intermediate results, and print as output straight-line programs. Let us first precisely define what we mean (see also [S1]).

DEFINITION. Let F be a field, $X = \{x_1, \ldots, x_n\}$ a set of indeterminates. Then $P = (X, V, C, S)$ is an *algebraic straight-line program* over $K = F(x_1, \ldots, x_n)$ if

(SLP1) $S = \{s_1, \ldots, s_k\} \subset F$, $V = \{v_1, \ldots, v_l\}$, $V \cap K = \varnothing$. X is called the sets of *inputs*, V the set of (program) *variables*, and S the set of *scalars*.

(SLP2) $C = (v_\lambda \leftarrow v'_\lambda \, \mathrm{O}_\lambda \, v''_\lambda)_{\lambda = 1, \ldots, l}$ with $\mathrm{O}_\lambda \in \{+, -, \times, \div\}$, $v'_\lambda, v''_\lambda \in S \cup X \cup \{v_1, \ldots, v_{\lambda - 1}\}$ for all $\lambda = 1, \ldots, l$. C is called the *computation sequence* and l the *length* of P, $l = \mathrm{len}(P)$.

(SLP3) For all $\lambda = 1, \ldots, l$ there exists $\mathrm{sem}(v_\lambda) \in K$, the *semantics* of v_λ, such that

$$\begin{aligned}
&\mathrm{sem}(a) = a \text{ if } a \in S \cup X,\\
&\mathrm{sem}(v_\lambda) = \mathrm{sem}(v'_\lambda) \pm \mathrm{sem}(v''_\lambda) \text{ if } \mathrm{O}_\lambda = \pm,\\
&\mathrm{sem}(v_\lambda) = \mathrm{sem}(v'_\lambda)\, \mathrm{sem}(v''_\lambda) \text{ if } \mathrm{O}_\lambda = \times,\\
&\mathrm{sem}(v''_\lambda) \neq 0 \text{ and } \mathrm{sem}(v_\lambda) = \mathrm{sem}(v'_\lambda)/\mathrm{sem}(v''_\lambda) \text{ if } \mathrm{O}_\lambda = \div.
\end{aligned}$$

The *set of elements* computed by P is $\mathrm{sem}(P) = \bigcup_{\lambda = 1}^{l} \{\mathrm{sem}(v_\lambda)\}$. □

We say $f \in F[x_1, \ldots, x_n]$ is *given by* the straight-line program $P = (X, V, C, S)$ if $f \in \mathrm{sem}(P)$. Notice that we use the notation

$f \in \text{sem}(P)$ with the implied understanding that we also know the $v_\lambda \in V$ with $f = \text{sem}(v_\lambda)$. Straight-line programs are originally meant to be evaluated at points $\phi(x_i) \in F$. It can happen that such an evaluation is impossible due to a division by zero. We say that P is *defined* at $\phi: \{x_1, \ldots, x_n\} \to F$ if a division by zero does not occur during evaluation of P at $\phi(x_i)$ in place of x_i, $1 \leqslant i \leqslant n$.

Here we will not describe a concrete data structure that can be used to represent straight-line programs on an algebraic RAM. It is fairly easy to conceive of suitable ones, e.g., labeled directed acyclic multigraphs can be used. A more intricate data structure was used for the first implementation of our algorithm and is described in [FI]. At this point it is convenient to define the *element size* of a straight-line program as

$$\text{el-size}(P) = \sum_{v_\lambda^* \in X \cup S, * \in \{',''\}} \text{size}(v_\lambda^*).$$

Notice that the actual size of P is in bits

$$O(\text{len}(P) \log \text{len}(P) + \text{el-size}(P)),$$

since it takes $\text{size}(v_\lambda) = O(\log(\lambda))$ bits to represent v_λ in address memory.

We now produce the input and output specifications of those algorithms presented in [K7], which we will need for the algorithms discussed in this chapter.

3.1. Algorithm *Zero-Division Test*

Input. A straight-line program $P = (\{x_1, \ldots, x_n\}, V, C, \{s_1, \ldots, s_m\})$ of length l over $\mathbf{Q}(x_1, \ldots, x_n)$, $a_\nu \in \mathbf{Q}$, $1 \leqslant \nu \leqslant n$, and a failure probability $\varepsilon \ll 1$.

Output. An integer p such that P is defined at ψ with $\psi(x_\nu) = a_\nu \bmod p$, $\psi(s_\mu) = s_\mu \bmod p$, or "failure." The latter occurs with probability $< \varepsilon$ in case P is defined at ϕ given by $\phi(x_\nu) = a_\nu$. □

3.2. Algorithm *Evaluation*

Input. As in algorithm Zero-Division Test. Furthermore an index λ, $1 \leqslant \lambda \leqslant l$, and a bound B_λ.

Output. Either "failure" (that with probability $< \varepsilon$ in case P is defined at ϕ) or $e_\lambda = \text{sem}(\phi(v_\lambda))$ provided that

$$|\text{numerator}(e_\lambda)|, |\text{denominator}(e_\lambda)| \leqslant B_\lambda. \qquad \square$$

Both algorithms have a binary complexity of order $(l \log(B) \log(1/\varepsilon)^{O(1)})$ on a probabilistic algebraic RAM over $\mathbf{Q}$, where $B = \max(\text{size}(a_\nu), \text{size}(s_\mu), B_\lambda)$ [K7].

3.3. Algorithm *Polynomial Coefficients*

Input. $f \in F[x_1, \ldots, x_n]$ given by a straight-line program $P = (\{x_1, \ldots, x_n\}, V, C, S)$ over $F(x_1, \ldots, x_n)$ of length l, a failure probability $\varepsilon \ll 1$, and a bound $d \geqslant \deg_{x_1}(f)$.

Output. Either "failure," this with probability $> \varepsilon$, or a straight-line program $Q = (\{x_2, \ldots, x_n\}, V_Q, C_Q, S_Q)$ over $F(x_2, \ldots, x_n)$ such that

$$\{c_0, \ldots, c_d\} \subset \text{sem}(Q) \quad \text{and} \quad \text{len}(Q) = O(ld + M(d) \log d),$$

where $c_\delta \in F[x_2, \ldots, x_n]$ satisfies

$$f = \sum_{\delta=0}^{d} c_\delta(x_2, \ldots, x_n) x_1^\delta.$$

Here and later $M(d)$ denotes a function dominating the time for multiplying polynomials in $F[x]$ of maximum degree d. Notice that for arbitrary fields the best known upper bound for $M(d)$ is $O(d \log(d) \log\log(d))$ [Sh1]. $\square$

The running time of this algorithm is summarized by the following theorem, which is typical for our theory.

THEOREM 3.1. Algorithm Polynomial Coefficients does not fail with probability $> 1 - \varepsilon$. It requires polynomially many arithmetic steps in d and l on a probabilistic algebraic RAM over F. For $F = \mathbf{Q}$ and $F = \mathbf{F}_q$ its binary complexity is also polynomial in el-size(P) and $\log(1/\varepsilon)$ [K7], Theorem 5.1. $\square$

The Polynomial Coefficients algorithm requires the knowledge of a bound $d \geqslant \deg_{x_1}(f)$. If no such bound is given, we can probabilistically guess the degree by running our algorithm for

$$d = 1, 2, 4, \ldots, 2^k, \cdots .$$

Let $f_k(x_1, \ldots, x_n)$ be the interpolation polynomial that is produced for the kth run. We then choose $a_1, \ldots, a_n \in R$ randomly and probabilistically test whether

$$f(a_1, \ldots, a_n) - f_k(a_1, \ldots, a_n) = 0.$$

This test can be performed by a simple modification of the Zero-Division Test algorithm, and the chance that the difference is falsely determined as 0 can be made smaller than ε. The probability that the randomly selected a_i certify the inequality of f and f_k can by Eq. (1) be made exponentially close to 1. Of course, by further testing $c_\delta(x_2, \ldots, x_n)$ for zero, $\delta = 2^k, 2^k - 1, \ldots$ we can get a probabilistic estimate for the actual degree $\deg_{x_1}(f)$. This procedure has expected polynomial running time in $\deg_{x_1}(f)$, and can be made quite efficient by computing the f_k incrementally [FI]. The total degree of f can be similarly estimated by testing $\deg_{x_1}(\tilde{f})$, where

$$\tilde{f}(x_1, \ldots, x_n) = f(x_1, x_2 + b_2 x_1, \ldots, x_n + b_n x_1)$$

with b_i randomly selected [K7, Lemma 5.1]. A similar algorithm is also described in [G2, Remark 5.4]. More general, one can even probabilistically determine the degrees of the numerator and denominator of a rational function computed by the input program, and therefore one can probabilistically test whether it computes a polynomial to start with, see [K8], Corollary 4.1.

4. CONVERSION INTO SPARSE POLYNOMIALS

We now discuss our version of Zippel's [Zi1] sparse interpolation algorithm for converting a polynomial from its straight-line to its sparse representation. The *sparse representation* for

$$f(x_1, \ldots, x_n) = \sum_{(e_1, \ldots, e_n) \in J} c_{e_1, \ldots, e_n} x_1^{e_1} \cdots x_n^{e_n},$$

$$0 \neq c_{e_1, \ldots, e_n} \in F, \qquad J \subset \mathbf{N}^n$$

is the vector

$$((e_1, \ldots, e_n, c_{e_1,\ldots,e_n}))_{(e_1,\ldots,e_n)\in J}.$$

Here $\mathbf{N}$ denotes the set of nonnegative integers. We write $\mathrm{mon}(f) = \mathrm{card}(J)$, the number of monomials in f, and $\mathrm{supp}(f) = J$, the *support* or set of nonzero monomial exponents of f. Zippel's algorithm is based on the idea that during the variable by variable interpolation process any zero coefficient is, with high probability, the image of a zero polynomial. We first present the algorithm for general fields. Extra difficulties arising from coefficient size growth are dealt with afterwards.

4.1. Algorithm *Sparse Conversion*

Input. $f \in F[x_1, \ldots, x_n]$ given by a straight-line program P of length l. Furthermore, a bound $d_0 \geqslant \max_{1\leqslant i\leqslant n}\{\deg_{x_i}(f)\}$, the allowed failure probability $\varepsilon \ll 1$, and an upper bound $t \leqslant (d_0+1)^n$ for the number of monomials permitted in the answer.

Output. Either "failure" (that with probability $< \varepsilon$), or the representation of a sparse polynomial with no more than t monomials, or the message "f has (probably) more than t monomials." The latter two outputs are correct with probability $> 1 - \varepsilon$.

Step R (Select Initial Evaluation Points): From a set $R \subset F$ with

$$\mathrm{card}(R) > \frac{1}{\varepsilon}\max\,(n\deg(f)(d_0+1)^n,\ (n(d_0+1)t+1)2^{l+1} + n\deg(f)(d_0+1)t)$$

select random elements $a_2, \ldots, a_n \in R$. Notice that if $\deg(f)$ is not known one can use the crude upper bound $\deg(f) \leqslant nd_0$.

Step L (Interpolation Loop): **For** $i \leftarrow 1, \ldots, n$ **Do** Step I. Then return $\sum c_{e_1,\ldots,e_n} x_1^{e_1}\cdots x_n^{e_n}$, $c_{e_1,\ldots,e_n} \neq 0$.

Step I (Interpolate One More Variable): At this point we have with high probability correctly computed the sparse representation of

$$f(x_1, \ldots, x_{i-1}, a_i, \ldots, a_n) = \sum_{(e_1,\ldots,e_{i-1})\in J_i} c_{e_1,\ldots,e_{i-1}} x_1^{e_1}\cdots x_{i-1}^{e_{i-1}},$$

$$0 \neq c_{e_1,\ldots,e_{i-1}} \in F, \qquad J_i \subset \mathbf{N}^{i-1}.$$

For $i = 1$ we have $J_1 = \{\varnothing\}$. We need not know $c_\varnothing = f(a_1, \ldots, a_n)$. Set

$$j_i \leftarrow \operatorname{card}(\{(e_1, \ldots, e_{i-1}, \delta) \mid (e_1, \ldots, e_{i-1}) \in J_i,\ 0 \leqslant \delta \leqslant d_{e_1, \ldots, e_{i-1}}\}),$$

where $d_{e_1, \ldots, e_{i-1}} = \min[d_0, \deg(f) - e_1 - \cdots - e_{i-1}]$.

For $k \leftarrow 1, \ldots, j_i$ **Do**

From the subset R select random points $b_{k,1}, b_{k,2}, \ldots, b_{k,i} \in R$. Compute

$$g_{k,i} = f(b_{k,1}, \ldots, b_{k,i}, a_{i+1}, \ldots, a_n)$$

by evaluating P at $\phi_{k,i}(x_\mu) = b_{k,\mu}$, $1 \leqslant \mu \leqslant i$, $\phi_{k,i}(x_v) = a_v$, $i + 1 \leqslant v \leqslant n$. If P is not defined at $\phi_{k,i}$ return "failure."

Solve the j_i by j_i linear system

$$\sum_{(e_1, \ldots, e_{i-1}) \in J_i} \sum_{\delta=0}^{d_{e_1, \ldots, e_{i-1}}} \gamma_{e_1, \ldots, e_{i-1}, \delta} b_{k,1}^{e_1} \cdots b_{k,i-1}^{e_{i-1}} b_{k,i}^{\delta} = g_{k,i}, \quad 1 \leqslant k \leqslant j_i, \tag{2}$$

in the determinates $\gamma_{e_1, \ldots, e_{i-1}, \delta}$. If the system is singular, report "failure."

Set $c_{e_1, \ldots, e_i} = \gamma_{e_1, \ldots, e_i}$, where the right-hand side ranges over all nonzero components of the solution of the above system. Notice that the subscripts of these components define the set J_{i+1}. If the number of those nonzero coefficients becomes more than t, return "input polynomial has (probably) more than t monomials." □

The challenging part is the verification of the failure and incorrectness probabilities. For this, it is helpful to prove the following lemma.

LEMMA 4.1. Let $J_i \subset \mathbf{N}^i$, $\operatorname{card}(J_i) = j_i < \infty$. Then

$$\Delta_i = \det([\beta_{k,1}^{e_1} \cdots \beta_{k,i}^{e_i}]_{(e_1, \ldots, e_i) \in J_i;\, k = 1, \ldots, j_i})$$

is a nonzero polynomial in $F[\beta_{1,1}, \ldots, \beta_{j_i,i}]$.

Proof. Simply observe that the monomial contributed by the main diagonal of the determinant is unique. □

We now have the following theorem:

THEOREM 4.1. Algorithm Sparse Conversion does not fail and outputs the correct results with probability $> 1 - 2\varepsilon$. In that case it requires

$$O(n(ld_0t + d_0^3t^3))$$

arithmetic steps on a probabilistic algebraic RAM over a (sufficiently large) field F.

Proof. Each of the $j_i \leqslant (d_0 + 1)t$ evaluations in step I requires $O(l)$ arithmetic steps. Solving the j_i by j_i system takes $O(j_i^3)$ steps. Notice that this bound also includes setting up the linear system from J_i and $g_{k,i}$. Step I is executed n times, which shows the stated complexity.

We now analyze the probabilistic behaviour of the algorithm. Let us first assume that the algorithm does not fail. A correct answer is returned provided the system in Eq. (2) captures for all i every nonzero monomial coefficient of $f(x_1, \ldots, x_i, a_{i+1}, \ldots, a_n)$. Let

$$f(x_1, \ldots, x_i, \alpha_{i+1}, \ldots, \alpha_n) = \sum_{(e_1, \ldots, e_i) \in \hat{J}_{i+1}} \hat{c}_{e_1, \ldots, e_i} x_1^{e_1} \cdots x_i^{e_i},$$

$0 \neq \hat{c}_{e_1, \ldots, e_i} \in F[\alpha_{i+1}, \ldots, \alpha_n]$, and let

$$\sigma_i = \prod_{(e_1, \ldots, e_i) \in \hat{J}_{i+1}} \hat{c}_{e_1, \ldots, e_i}, \qquad \deg(\sigma_i) \leqslant \operatorname{mon}(f) \deg(f).$$

Notice that in general $\hat{J}_{i+1} \supseteq J_{i+1}$. But $\sigma_i(a_{i+1}, \ldots, a_n) \neq 0$ implies that $\hat{J}_{i+1} = J_{i+1}$, which in turn means that the unique solution to Eq. (2) must determine $f(x_1, \ldots, x_i, a_{i+1}, \ldots, a_n)$. Since $\operatorname{mon}(f) \leqslant (d_0 + 1)^n$ the probability

$$\operatorname{Prob}(\sigma_i(a_{i+1}, \ldots, a_n) \neq 0 \text{ for all } 1 \leqslant i \leqslant n)$$

is not less than

$$1 - \sum_{i=1}^{n} \frac{\deg(\sigma_i)}{\operatorname{card}(R)} \geqslant 1 - \frac{n \operatorname{mon}(f) \deg(f)}{\operatorname{card}(R)}$$

$$\geqslant 1 - \frac{n \deg(f)(d_0 + 1)^n}{\operatorname{card}(R)} > 1 - \varepsilon.$$

We now estimate the failure probability. We define the events

$$E_0 = \{(a_2, \ldots, a_n) \mid P \text{ is defined at } \phi_0(x_1) = x_1, \quad \phi_0(x_i) = a_i,\ 2 \leqslant i \leqslant n\}$$

and

$$E_{k,i} = \{(b_{k,1}, \ldots, b_{k,i}) \mid P \text{ is defined at } \phi_{k,i}\}.$$

As in [K7, Lemma 4.2], we have both

$$\text{Prob}(E_0),\ \text{Prob}(E_{k,i} \mid E_0) \geqslant 1 - \frac{2^{l+1}}{\text{card}(R)}.$$

Since $j_i \leqslant (d_0 + 1)t$ we get

$$\text{Prob}\left(E_0 \cap \bigcap_{\substack{i=1,\ldots,n \\ k=1,\ldots,j_i}} E_{k,i}\right) \geqslant \left(1 - \frac{2^{l+1}}{\text{card}(R)}\right) \prod_{\substack{i=1,\ldots,n \\ k=1,\ldots,j_i}} \left(1 - \frac{2^{l+1}}{\text{card}(R)}\right)$$

$$\geqslant 1 - \frac{(n(d_0 + 1)t + 1)2^{l+1}}{\text{card}(R)}.$$

Now by Lemma 4.1 for a given i the coefficient matrix for Eq. (2) is nonsingular with probability no less than

$$1 - \frac{\deg(\Delta_i)}{\text{card}(R)} \geqslant 1 - \frac{\deg(f) j_i}{\text{card}(R)} \geqslant 1 - \frac{\deg(f)(d_0 + 1)t}{\text{card}(R)}.$$

Thus all n arising systems are nonsingular with probability $\geqslant 1 - n\deg(f)(d_0 + 1)t/\text{card}(R)$. Therefore, the algorithm fails with probability no more than

$$\frac{1}{\text{card}(R)}\left((n(d_0 + 1)t + 1)2^{l+1} + n\deg(f)(d_0 + 1)t\right). \qquad \square$$

We wish to remark that the input parameters d_0 and t need not be specified beforehand. In Section 3 we have discussed how to probabilistically determine $d_i = \deg_{x_i}(f)$. In fact, the Sparse Conversion algorithm runs more efficiently if we use d_i in place of

d_0 for the ith iteration of step I. The parameter t is used only to abort execution in case f has too many monomials or that we are interpolating with unlucky evaluation points. By adjusting card(R) appropriately we can achieve expected polynomial running time also in mon(f) without the input parameter t. In the context of an actual computer algebra system we prefer our formulation of the algorithm, whose running time is independent of bad random choices.

We now discuss the complications arising for $F = \mathbf{Q}$. Our requirement is to accomplish binary polynomial-time complexity. It is clear that we must include the *coefficient size* of f,

$$\text{c-size}(f) = \max_{(e_1,\ldots,e_n)\in \text{supp}(f)} \{\text{size}(c_{e_1,\ldots,e_n})\},$$

into our input parameters. One might think that all we have to do is use the Evaluation algorithm of Section 3 inside the FOR loop of step I and adjust the failure probability accordingly. Unfortunately, there exists a theoretical probability that size $(g_{k,i})$ is exponential in n. This would happen, for instance, if all denominators of $c_{e_1,\ldots,e_n}$ were distinct primes and mon(f) were exponential in n. A way to overcome this problem is to perform the entire conversion modulo p, p an integer that has been tested to be a prime with probability $\geqslant 1 - \varepsilon$ [SS, R1], and retrieve the rational coefficients of f by a continued fraction approximation from the coefficients of $f \bmod p$ as in step C of the cited Evaluation algorithm. The pseudo-prime p must be selected such that also with probability $\leqslant 1 - \varepsilon$, P is defined at $\phi(x_v) = x_v$, $\phi(s) = s \bmod p$ for all $s \in S$ (see the Zero-Division Test algorithm cited in Section 3), and such that

$$p \geqslant \text{card}(R),\ 2^{2\text{c-size}(f)+1}.$$

In practice, it is better to work modulo $\bar{p}^k$ at the danger of increasing the failure probability. Then one avoids the generation of the rather large pseudoprime p, and one can also solve the linear system [Eq. (2)] $\bar{p}$-adically [FI]. For the record, let us state the following theorem.

THEOREM 4.2. For $F = \mathbf{Q}$, algorithm Sparse Conversion, if used in conjunction with a probabilistic primality test, the Zero-Division Test algorithm, and a recovery procedure for rational

numbers from their modular images, can complete and determine a correct answer with probability $\geqslant 1 - 3\varepsilon$. Its binary running time is polynomial in l, d_0, $\log(1/\varepsilon)$, t, el-size(P), and the additional input parameter that is a bound for c-size(f). □

An interesting result concerning the conversion of a straight-line program to a sparse rational function is a direct consequence of this theorem and the Numerator and Denominator algorithm in [K7].

COROLLARY 4.1. Let f/g be given by a straight-line program $P, f, g \in F[x_1, \ldots, x_n]$, $\mathrm{GCD}(f, g) = 1$, $d \geqslant \deg(f), \deg(g)$, $0 < \varepsilon \ll 1$. In order to avoid ambiguity assume that the coefficient of the lexicographically first monomial in g is 1. Provided the sparse representation of f, respectively g, has less than t monomials, it can be computed correctly with probability $> 1 - \varepsilon$ on a probabilistic algebraic RAM over F in polynomially many arithmetic steps in len(P), d, and t. In case $F = \mathbf{Q}$ the binary running time is also polynomial in el-size(P), $\log(1/\varepsilon)$, and c-size(f), respectively c-size(g). □

Before we can apply theorem 4.2 to the Polynomial GCD algorithm in [K7] we must introduce a slightly more restricted notion of coefficient size of f, that where the coefficients are already brought to a common denominator. Assume that

$$c_{e_1,\ldots,e_n} = \frac{u_{e_1,\ldots,e_n}}{u_*}, \qquad u_{e_1,\ldots,e_n}, u_* \in \mathbf{Z} \text{ for all } (e_1, \ldots, e_n) \in \mathrm{supp}(f).$$

Then the *combined coefficient size* of f is defined as

$$\text{cc-size}(f) = \text{size}(u_*) + \max_{(e_1,\ldots,e_n)\in \mathrm{supp}(f)} \{\text{size}(u_{e_1,\ldots,e_n})\}.$$

Now, since the size of the coefficients of integral multivariate polynomial factors can be polynomially bounded [Ge, Chapter III, §4, Lemma II], we obtain from the straight-line GCD algorithm in [K7] the following typical corollary.

COROLLARY 4.2. Let $f_\rho \in F[x_1, \ldots, x_n]$ be given by a straight-line program P, $d \geqslant \deg(f_\rho)$, $1 \leqslant \rho \leqslant r$, $g = \mathrm{GCD}_{1\leqslant\rho\leqslant r}(f_\rho)$, $0 < \varepsilon \ll 1$. Provided the sparse representation of g has less than t monomials,

it can be computed correctly with probability $> 1 - \varepsilon$ on a probabilistic algebraic RAM over F in polynomially many arithmetic steps in len(P), d, and t. In case $F = \mathbf{Q}$ the binary running time is also polynomial in el-size(P), $\log(1/\varepsilon)$, and $\min_{1 \leqslant \rho \leqslant r}\{\text{cc-size}(f_\rho)\}$. □

Notice that we cannot yet prove the above corollary for c-size(f_ρ) replacing cc-size(f_ρ). Therefore, one might question whether our restriction is reasonable. The answer is that for three large subclasses of polynomial representations, namely

Sparse polynomials, Formulas, and Determinants,

the combined coefficient size as well as the degrees are polynomially related to the input size. In fact, we know of no example for a straight-line program representing a polynomial of polynomially bounded degree and coefficient size, but where the combined coefficient size becomes exponential.

We shall concluce this section with a remark on counting the number of monomials. Clearly, the Sparse Conversion algorithm can probabilistically produce the number of monomials in a polynomial given by a straight-line program in time polynomial in the unary representation of that count. One may question whether it is possible to find the number of monomials in binary or random polynomial-time. This is most likely not the case due to the fact that the evaluation of 0–1 permanents is #P-hard [V1]. For if we replace all 1 entries in a 0–1 matrix by indeterminates $x_{i,j}$, i the corresponding row and j the corresponding column, then the number of monomials in the determinant of the new matrix is equal to the permanent of the original 0–1 matrix. Therefore the problem of counting the number of monomials in families of polynomials with polynomially bounded degree and straight-line computation length, which Valiant calls p-computable [V2], is #P-hard.

5. EVALUATION AND FACTOR DEGREE PATTERN

It is crucial for our Factoring algorithm that the Hensel lifting is started with true factor images. Fortunately, the effective versions of the Hilbert irreducibility theorem [G2, K4] make it possible to probabilistically enforce this assumption. In this section we present

a theorem (Theorem 5.2) on the probabilities that certain evaluations preserve the *factor degree pattern* that determines the number of irreducible factors, their multiplicities, and their total degrees. The argument is essentially the same as that in [G2, Theorem 3.6], but with our effective version of the Hilbert irreducibility theorem (Theorem 5.1). The main advantage of this change is that the evaluations are simpler and the probability of success is higher.

THEOREM 5.1 (*Effective Hilbert Irreducibility Theorem*). Let $f(x_1, \ldots, x_n) \in F[x_1, \ldots, x_n]$, F a field, have total degree d and be irreducible. Furthermore, assume that x_2 occurs in f, that is $\deg_{x_2}(f) > 0$. If $\text{char}(F) = p > 0$ we require that each coefficient of f in F possesses a pth root in F. A sufficient condition for this to be true is that F be perfect. Let $R \subset F$ and let $a_1, a_3, \ldots, a_n$, $b_3, \ldots, b_n$, be random elements in R. Then the probability

$$\text{Prob}(f(x_1 + a_1, x_2, b_3 x_1 + a_3, \ldots, b_n x_1 + a_n) \text{ becomes reducible in } F[x_1, x_2]) \leqslant \frac{4d2^d}{\text{card}(R)}.$$

For a proof see [K4, Theorem 3]. □

In the following association between two polynomials f and g is denoted by $f \sim g$ and means that $f = cg$ with $0 \neq c \in F$. The factor degree pattern of $f \in F[x_1, \ldots, x_n]$ is defined as a lexicographically ordered vector $[(d_i, e_i)]_{i=1,\ldots,r}$ such that for $f = \prod_{i=1}^{r} h_i^{e_i}$, $h_i \in F[x_1, \ldots, x_n]$,

h_i irreducible, $d_i = \deg(h_i) \geqslant 1$, $1 \leqslant i \leqslant r$, $h_i \nsim h_j$, $1 \leqslant i \neq j \leqslant r$.

We want to apply Theorem 5.1 to the irreducible factors of a multivariate polynomial. However, Theorem 5.1 will apply only to those factors that depend on x_2. Therefore we need the following notion. The *primitive part* of a polynomial with respect to a variable is the polynomial divided by the GCD of all (polynomial) coefficients of that variable. We denote it by $\text{pp}_x(\,.\,)$, where x is the corresponding variable. In particular, if no factors are independent of x we call the polynomial primitive in x.

THEOREM 5.2. Let $f \in F[x_1, \ldots, x_n]$, F a perfect field, $d = \deg(f)$, $R \subset F$. Let $a_1, a_3, \ldots, a_n, b_3, \ldots, b_n \in R$ be randomly selected elements,

$$f_2 = f(x_1 + a_1, x_2, b_3 x_1 + a_3, \ldots, b_n x_1 + a_n).$$

Then

$$\text{Prob}(\text{pp}_{x_2}(f) \text{ and } \text{pp}_{x_2}(f_2) \text{ have the same factor degree pattern}) \geqslant 1 - \frac{4d2^d + d^3}{\text{card}(R)}.$$

Proof. First we consider all the factors h_i with $\deg_{x_2}(h_i) > 0$. By Theorem 5.1,

$$h_{i,2} = h_i(x_1 + a_1, x_2, b_3 x_1 + a_3, \ldots, b_n x_1 + a_n)$$

remains irreducible in $F[x_1, x_2]$ with probability $\geqslant 1 - 4d_i 2^{d_i}/\text{card}(R)$. It remains to estimate the probability that $\deg(h_{i,2}) = d_i$ and that $h_{i,2} \sim h_{j,2}$ for all $j \neq i$. Let

$$\hat{h}_i(x_1, x_2, \alpha_1, \alpha_3, \ldots, \alpha_n, \beta_3, \ldots, \beta_n) = h_i(x_1 + \alpha_1, x_2, \beta_3 x_1 + \alpha_3, \ldots, \beta_n x_1 + \alpha_n),$$

$\hat{h}_i \in F[x_1, x_2, \alpha_1, \alpha_3, \ldots, \alpha_n, \beta_3, \ldots, \beta_n]$. Clearly, $\deg_{x_1, x_2}(\hat{h}_i) = d_i$. Let

$$0 \neq \pi_i(\beta_3, \ldots, \beta_n) \in F[\beta_3, \ldots, \beta_n]$$

be the coefficient of a monomial $x_1^{j_1} x_2^{j_2}$, $j_1 + j_2 = d_i$, in $\hat{h}_i$. Now $\deg(\pi_i) \leqslant d_i$ and

$$\pi_i(b_3, \ldots, b_n) \neq 0 \text{ implies } \deg(h_{i,2}) = d_i.$$

By [Sw], Lemma 1, this happens with probability no less than

$$1 - \frac{\deg(\pi_i)}{\text{card}(R)} \geqslant 1 - \frac{d_i}{\text{card}(R)}.$$

We finally estimate the chance that $h_{i,2} \sim h_{j,2}$. First we claim that $\hat{h}_i \sim \hat{h}_j$, $i \neq j$, in $\bar{F}[x_1, x_2]$, $\bar{F} = F(\alpha_1, \alpha_3, \ldots, \alpha_n, \beta_3, \ldots, \beta_n)$. For if

this were not the case, then there would exist nonzero $g_i, g_j \in F[\alpha_1, \alpha_3, \ldots, \alpha_n, \beta_3, \ldots, \beta_n]$, $\mathrm{GCD}(g_i, g_j) = 1$, such that $(g_i/g_j)\hat{h}_i = \hat{h}_j$. Hence either one of $\hat{h}_i$ or $\hat{h}_j$ would have to be reducible in $F[x_1, x_2, \alpha_1, \alpha_3, \ldots, \alpha_n, \beta_3, \ldots, \beta_n]$, say $\hat{h}_i = \hat{h}_i^{(1)}\hat{h}_i^{(2)}$. However then

$$h_i = (\hat{h}_i^{(1)}\hat{h}_i^{(2)})(x_1 - \alpha_1, x_2, \alpha_1, x_3 - \beta_3(x_1 - \alpha_1), \ldots, x_n - \beta_n(x_1 - \alpha_1), \beta_3, \ldots, \beta_n)$$

would be nontrivial factorization of h_i that would necessarily have to lie in $F[x_1, \ldots, x_n]$, in contradiction to the irreducibility of h_i. This shows nonassociatness of $\hat{h}_i$ and $\hat{h}_j$ over $\bar{F}$. We now have two coefficients of $\hat{h}_i$ in $\bar{F}$, that is

$$\hat{h}_i = \cdots + \sigma_i^{(\lambda_1, \lambda_2)} x_1^{\lambda_1} x_2^{\lambda_2} + \cdots + \sigma_i^{(\mu_1, \mu_2)} x_1^{\mu_1} x_2^{\mu_2} + \cdots,$$
$$\sigma_i^{(\lambda_1, \lambda_2)}, \sigma_i^{(\mu_1, \mu_2)} \in \bar{F},$$

and two corresponding coefficients in $\hat{h}_j$,

$$\hat{h}_j = \cdots + \sigma_j^{(\lambda_1, \lambda_2)} x_1^{\lambda_1} x_2^{\lambda_2} + \cdots + \sigma_j^{(\mu_1, \mu_2)} x_1^{\mu_1} x_2^{\mu_2} + \cdots,$$
$$\sigma_j^{(\lambda_1, \lambda_2)}, \sigma_j^{(\mu_1, \mu_2)} \in \bar{F},$$

such that

$$\tau_{i,j} = \sigma_i^{(\lambda_1, \lambda_2)} \sigma_j^{(\mu_1, \mu_2)} - \sigma_i^{(\mu_1, \mu_2)} \sigma_j^{(\lambda_1, \lambda_2)} \neq 0.$$

Now $\tau_{i,j} \in F[\alpha_1, \alpha_3, \ldots, \alpha_n, \beta_3, \ldots, \beta_n]$ and it is relatively easy to see that

$$\tau_{i,j}(a_1, a_3, \ldots, a_n, b_3, \ldots, b_n) \neq 0 \text{ implies } h_{i,2} \sim h_{j,2}.$$

Since $\deg(\tau_{i,j}) \leqslant d_i + d_j$ the probability of this event is $\geqslant 1 - (d_i + d_j)/\mathrm{card}(R)$.

Now we consider those h_i with $\deg_{x_2}(h_i) = 0$. All that must be satisfied for the theorem to hold is that $h_{i,2}$ as defined above is not identical zero. Again the total degree of h_i gets preserved with probability $d_i/\mathrm{card}(R)$, which is a sufficient condition. Overall,

the factor degree pattern is preserved with probability not less than

$$1 - \left(\sum_{i=1}^{r} \frac{4d_i 2^{d_i}}{\operatorname{card}(R)} + \sum_{i=1}^{r} \frac{d_i}{\operatorname{card}(R)} + \sum_{1 \leqslant i < j \leqslant r} \frac{d_i + d_j}{\operatorname{card}(R)} \right)$$

$$\geqslant 1 - \left(\frac{4d2^d}{\operatorname{card}(R)} + \frac{d}{\operatorname{card}(R)} + \frac{d(d-1)}{2} \frac{d}{\operatorname{card}(R)} \right)$$

$$\geqslant 1 - \frac{4d2^d + d^3}{\operatorname{card}(R)}. \qquad \square$$

One can probabilistically enforce that the input polynomial is primitive in x_2 by making the linear substitution $x_i + c_i x_2$ for all x_i, $i \neq 2$, with randomly chosen c_i. This substitution does not affect the factor degree pattern. It should be clear from the above theorem that we thus can probabilistically obtain the factor degree pattern of a polynomial given by a straight-line program by evaluation. We formulated Theorem 5.2 in its generality because we will make a slightly different substitution in the Factorization algorithm in Section 6. Moreover, the theorem in its current form can be used to also compute the degrees of individual variables in the factors. One lets each variable take the role of x_2 and identifies the factors in the different bivariate domains by evaluating that variable at a linear form. However, since this result is not needed in the following, we shall skip the details.

The assumption that the field is perfect can be dropped at the cost of increasing the failure probability somewhat (see [G2, Lemma 4.2]), but since the usual coefficient fields are perfect we do not incorporate this generalization.

6. STRAIGHT-LINE FACTORIZATION

We now describe the algorithm for finding the straight-line factors of a straight-line polynomial. The algorithm is derived from the One-Variable Lifting algorithm in [K5], with the homogeneous parts of the minor variables replacing the monomials of the single variable with respect to which is lifted. Note that a homogeneous polynomial of degree d has the form

$$\sum_{e_1 + \cdots + e_n = d} c_{e_1, \ldots, e_n} x_1^{e_1} \cdots x_n^{e_n}, \qquad c_{e_1, \ldots, e_n} \in F.$$

We will compute those homogeneous parts by straight-line programs. The main reason why the answer is polynomial in length is that we only need to add on to the intermediate programs. This is because subsequent homogeneous parts can be computed from previous ones and Strassen's technique of obtaining a homogeneous program need not be applied at each iteration.

6.1. Algorithm *Factorization*

Input. $f \in F[x_1, \ldots, x_n]$ given by a straight-line program P of length l, a bound $d \geqslant \deg(f)$, and an allowed failure probability $\varepsilon \ll 1$.

Output. Either "failure," that with probability $< \varepsilon$, or $e_i \geqslant 1$ and irreducible $h_i \in F[x_1, \ldots, x_n]$, $1 \leqslant i \leqslant r$, given by a straight-line program Q of length

$$\operatorname{len}(Q) = O(d^2 l + dM(d^2)\log(d))$$

such that with probability $> 1 - \varepsilon$, $f = \prod_{i=1}^{r} h_i^{e_i}$. [Refer to algorithm Polynomial Coefficients in Section 3 for the definition of $M(.)$.] In case $p = \operatorname{char}(F)$ divides any e_i, that is $e_i = p^{\hat{e}_i}\bar{e}_i$ with $\bar{e}_i$ not divisible by p, we return $\bar{e}_i$ in place of e_i and Q will compute $h_i^{p^{\hat{e}_i}}$.

Step R (Random Points Selection): From a set $R \subset F$ with

$$\operatorname{card}(R) > \frac{6}{\varepsilon}\max(2^{l+2}, d2^d + d^3, 2(d+1)^4)$$

select random elements $a_1, \ldots, a_n, b_2, \ldots, b_n, c_1, c_3, \ldots, c_n$. If $F = \mathbf{F}_q$ with q small we can instead work over $\mathbf{F}_{q^p}$, p a prime integer $> d$. By Theorem 6.1 in [G2] no additional factors occur.

Test whether P is defined at $\phi(x_i) = a_i$, $1 \leqslant i \leqslant n$. For $F = \mathbf{Q}$ we call algorithm Zero-Division Test in [K7] such that the probability of "failure" even if P were defined at ϕ is less than $\varepsilon/6$. If P turns out to be (probably) undefined at ϕ we return "failure." Otherwise, P is definitely defined as ϕ and we compute the dense representation of

$$f_2 = f(x_1 + c_1x_2 + a_1, x_2 + b_2x_1 + a_2, b_3x_1 + c_3x_2 + a_3, \ldots, b_nx_1 + c_nx_2 + a_n).$$

This can be done by evaluation and interpolation similarly to the Sparse Conversion algorithm. If $F = \mathbf{Q}$, a bound for the cc-size(f) must be added to the input parameters and we must again make the probability that "failure" occurs due to the use of modular arithmetic during evaluation less than $\varepsilon/6$.

Step F (Factorization): Factor

$$f_2 = \prod_{i=1}^{r} \tilde{g}_{i,2}^{e_i},$$

$\tilde{g}_{i,2} \in F[x_1, x_2]$ irreducible and pairwise not associated. Notice that by Theorem 5.1 f and f_2 have with high probability the same factor degree pattern, that is irreducible factors of f map to pairwise nonassociated irreducible factors of f_2 of the same total degrees. For the remainder of the algorithm we will assume that this is the case.

If $\text{char}(F) = p > 0$ divides any of the e_i, say $e_i = p^{\hat{e}_i}\bar{e}_i$ with $\bar{e}_i$ not divisible by p, we replace e_i by $\bar{e}_i$ and $\tilde{g}_{i,2}$ by $\tilde{g}_{i,2}^{p^{\hat{e}_i}}$. This replacement guarantees that none of the multiplicities is divisible by the characteristic. Now set

$$g_{i,0}(x_1) \leftarrow \tilde{g}_{i,2}(x_1, 0) \in F[x_1].$$

Check whether $\text{GCD}(g_{i,0}, g_{j,0}) \sim 1$ for $1 \leqslant i < j \leqslant r$ and whether $\deg(\tilde{g}_{i,2}) = \deg(g_{i,0})$ for $1 \leqslant i \leqslant r$. If not return "failure." Let

$$\begin{aligned} \bar{f}(x_1, \ldots, x_n) &= f(x_1 + a_1, x_2 + b_2 x_1 + a_2, \ldots, x_n + b_n x_1 + a_n) \\ &= \prod_{i=1}^{r} h_i(x_1, \ldots, x_n)^{e_i}, \end{aligned}$$

and assume that h_i are the factors that correspond to $\tilde{g}_{i,2}$. Notice that the assumptions on the preservation of the total degrees of the factors throughout the evaluation process also imply that

$$\text{ldcf}_{x_1}(\bar{f}) \in F. \tag{3}$$

Here ldcf_{x_1} denotes the coefficient of the highest power of x_1 and is generally a polynomial in $F[x_2, \ldots, x_n]$. Furthermore, let $\bar{P}$ be a straight-line program computing $\bar{f}$. We write

$$\bar{f}(x_1, \ldots, x_n) = \sum_{j=0}^{d} \sum_{m=0}^{d} \bar{f}_{j,m}(x_2, \ldots, x_n) x_1^m.$$

where $\bar{f}_{j,m} \in F[x_2, \ldots, x_n]$ is homogeneous of degree j. We remark that d can now be set to $\deg(f)$ rather than a bound for it. We will need a straight-line program that computes $\bar{f}_{j,m}$. If we replace x_i by $x_i x_1^{d+1}$, $2 \leqslant i \leqslant n$, in $\bar{P}$ then $\bar{f}_{j,m}$ is the coefficient of $x_1^{j(d+1)+m}$. Therefore by evaluating at x_1 and interpolating as in the Polynomial Coefficients algorithm (Section 3) we can find a straight-line program Q_0 for $\bar{f}_{j,m}$ of length

$$\text{len}(Q_0) = O(d^2 l + M(d^2)\log(d)).$$

Notice that we need to randomly pick $(d+1)^2$ distinct points at which we interpolate and we must make sure that the straight-line program $\bar{P}$ is defined at those points. If that is not the case, or if for $F = \mathbf{Q}$ we cannot confirm by the Zero-Division Test algorithm (Section 3) that a point is good, that with probability $< \varepsilon/(6(d+1)^2)$, we return "failure." For more details we refer to step P in the cited Polynomial Coefficients algorithm.

Step H (Hensel Lifting Loop): **For** $k \leftarrow 0, \ldots, d-1$ **Do** step L.

Step L (Lift by One Degree): Let

$$h_i(x_1, \ldots, x_n) = \sum_{m=0}^{d_i} \sum_{j=0}^{d_i} c_{i,j,m}(x_2, \ldots, x_n) x_1^m, \qquad d_i = \deg(h_i),$$

where $c_{i,j,m}(x_2, \ldots, x_n) \in F[x_2, \ldots, x_n]$ is homogeneous of degree j. At this point we have a straight-line program Q_k over $F(x_2, \ldots, x_n)$ that computes $c_{i,j,m}$ for $1 \leqslant i \leqslant r$, $0 \leqslant j \leqslant k$, $0 \leqslant m \leqslant d_i$, and all $\bar{f}_{j,m}$, $0 \leqslant j, m \leqslant d$. Notice that $c_{i,0,m} \in F$ is the coefficient of x_1^m in $g_{i,0}$ found in step F, and therefore Q_0 need not encode them. Whenever reference to these coefficients is made later, we just encode them as scalars. Notice also that by (3) $c_{i,j,d_i} = 0$ for $j > 0$. We will extend Q_k to Q_{k+1} that also computes $c_{i,k+1,m}$. It is useful to introduce the following polynomials

$$g_{i,k} = \sum_{j=0}^{k} \sum_{m=0}^{d_i} c_{i,j,m} x_1^m, \qquad \hat{g}_{i,k+1} = \sum_{m=0}^{d_i} c_{i,k+1,m} x_1^m.$$

Now consider the congruence

$$\prod_{i=1}^{r} (g_{i,k} + \hat{g}_{i,k+1})^{e_i} \equiv \bar{f} \bmod (x_2, \ldots x_n)^{k+2}. \tag{4}$$

Expanding the left-hand side we get

$$g_{1,0}^{e_1-1}\cdots g_{r,0}^{e_r-1}\sum_{i=1}^{r}\left(e_i\hat{g}_{i,k+1}\prod_{j=1}^{r}g_{j,0}\right)$$

$$\equiv \bar{f}-\prod_{i=1}^{r}g_{i,k}^{e_i} \bmod (x_2,\ldots,x_n)^{k+2}. \tag{5}$$

By our loop invariant for Q_k

$$\left(\bar{f}-\prod_{i=1}^{r}g_{i,k}^{e_i}\right) \bmod (x_2,\ldots,x_n)^{k+2}$$

$$= t_{k+1} = \sum_{m=0}^{d-1} t_{k+1,m}(x_2,\ldots,x_n)x_1^m,$$

where $t_{k+1,m}\in F[x_2,\ldots,x_n]$ is homogeneous of degree $k+1$. Notice that the degree of t_{k+1} in x_1 is $\leqslant d-1$ by assumption (3). We need a program T_{k+1} that computes $t_{k+1,m}$. However, T_{k+1} does not start from scratch, but references the program variables in Q_k that compute $c_{i,j,m}$ and $\bar{f}_{j,m}$. If T_{k+1} encodes a tree-like bivariate multiplication scheme with those program variables as undetermined coefficients, that can be done in

$$\operatorname{len}(T_{k+1}) = O(M(d^2)\log(d)).$$

Now, since t_{k+1} equals the left-hand side of Eq. (5) $g_{1,0}^{e_1-1}\cdots g_{r,0}^{e_r-1}$ must divide t_{k+1} in $F[x_1,\ldots,x_n]$. Notice that this claim might not be valid if $g_{i,0}$ is not an image of h_i, since then the existence of the $\hat{g}_{i,k+1}$ cannot be guaranteed. However, in that case our construction still completes, but the resulting straight-line answer is incorrect. Let

$$u_{k+1} = \sum_{m=0}^{d_1+\cdots+d_r-1} u_{k+1,m}x_1^m = \frac{t_{k+1}}{g_{1,0}^{e_1-1}\cdots g_{r,0}^{e_r-1}},$$

$$u_{k+1,m}\in F[x_2,\ldots,x_n].$$

Again, we need a straight-line program U_{k+1} that computes all $u_{k+1,m}$ from the program variables for $t_{k+1,m}$ in T_{k+1} as indeterminates. Since the leading coefficient in x_1 of $g_{1,0}^{e_1-1}\cdots g_{r,0}^{e_r-1}$ is an element in F, the $u_{k+1,m}$ can be determined by simply encoding a univariate

polynomial division in x_1 over the coefficient field $F(x_2, \ldots, x_n)$. Therefore we can construct U_{k+1} of length $\text{len}(U_{k+1}) = O(M(d))$. (Actually, the entire divisor is in $F[x_1]$ but our argument here also applies to a quadratic lifting procedure, see the remark below the proof of Theorem 6.1.) Now consider

$$\frac{u_{k+1}}{g_{1,0} \cdots g_{r,0}} = \frac{e_1 \hat{g}_{1,k+1}}{g_{1,0}} + \cdots + \frac{e_r \hat{g}_{r,k+1}}{g_{r,0}}.$$

It is clear that $e_i c_{i,j,k+1}$ are the coefficients of the univariate partial fraction decomposition of $u_{k+1}/(g_{1,0} \cdots g_{r,0})$ carried out over the field $F(x_2, \ldots, x_n)$. One way to compute these coefficients by a straight-line program $\hat{Q}_{k+1}$ with $\text{len}(\hat{Q}_{k+1}) = O(d^2)$ is to once and for all find $\hat{g}_{i,0}^{(m)} \in F[x_1]$, $0 \leqslant m \leqslant d_1 + \cdots + d_r - 1$, with

$$\frac{x_1^m}{g_{1,0} \cdots g_{r,0}} = \frac{\hat{g}_{1,0}^{(m)}}{g_{1,0}} + \cdots + \frac{\hat{g}_{r,0}^{(m)}}{g_{r,0}}, \qquad \deg(\hat{g}_{i,0}^{(m)}) < d_i,$$

and encode the summation

$$\hat{g}_{i,k+1} = \frac{1}{e_i} \sum_{m=0}^{d_1 + \cdots + d_r - 1} u_{k+1,m} \hat{g}_{i,0}^{(m)}, \qquad 1 \leqslant i \leqslant r.$$

We must be able to divide by e_i and here we need the fact that the multiplicities must not be divisible by $\text{char}(F)$. We finally link the programs Q_k, T_{k+1}, U_{k+1}, and $\hat{Q}_{k+1}$ properly together to obtain the program Q_{k+1}. Notice that

$$\text{len}(Q_{k+1}) \leqslant \text{len}(Q_k) + CM(d^2) \log(d),$$

where C is an absolute constant. Therefore $\text{len}(Q_{k+1}) = \text{len}(Q_0) + O((k+1)M(d^2) \log(d))$.

Step T (Final Translation): From Q_d we obtain Q that computes

$$h_i(x_1 - a_1, x_2 - b_2(x_1 - a_1) - a_2, \ldots, x_n - b_n(x_1 - a_1) - a_n)$$

by adding in front of Q_d instructions for translating the x_i appropriately. □

The following theorem summarizes the complexity of the above algorithm.

THEOREM 6.1. Algorithm Factorization does not fail with probability $> 1 - \varepsilon$. In that case it reduces the problem in polynomially many steps on a probabilistic algebraic RAM over F as a function in l and d to factoring bivariate polynomials. Its answer will be correct with probability $> 1 - \varepsilon$. It requires polynomially many randomly selected field elements. For $F = \mathbf{Q}$ or $F = \mathbf{F}_q$ the algorithm has binary polynomial complexity also in $\log(1/\varepsilon)$, el-size(P), and cc-size(f).

Proof. The arithmetic and binary running time is polynomial as a direct consequence of the results in [K7], in particular Theorems 3.1, 4.1, and 5.1. It remains to analyze the failure probabilities of the Factorization algorithm. The only way an incorrect program Q can be produced is that the factor degree of patterns of f and f_2 disagree. If $\deg_{x_2}(f) = \deg(f)$, which is true with probability $> 1 - d/\text{card}(R) > 1 - \varepsilon/12$, then by Theorem 5.2 this happens with probability less than

$$\frac{4d2^d + d^3}{\text{card}(R)} < \frac{4\varepsilon}{6}.$$

Thus the compound probability of getting an incorrect result is $< \varepsilon$.

"Failure" can occur in six separate circumstances. First, P may be undefined in ϕ, that with probability $< 2^{l+1}/\text{card}(R) < \varepsilon/6$ by Lemma 4.2 of [K7]. Second, for $F = \mathbf{Q}$ we might fail to recognize that P is defined at ϕ, but we make this possibility happen with probability $< \varepsilon/6$. Third, for $F = \mathbf{Q}$ the computation of f_2 may fail with probability $< \varepsilon/6$.

Fourth, "failure" can occur if for some $i \neq j$, $\text{GCD}(g_{i,0}, g_{j,0}) \sim 1$, or $\deg(g_{i,0}) < \deg(\tilde{g}_{i,2})$. Let $\pi_i(\beta_2) = \text{ldcf}_{x_1}(\tilde{g}_{i,2}(x_1, \beta_2 x_1 + \alpha_2))$ and let

$$\sigma_{i,j}(\alpha_2, \beta_2) = \text{resultant}_{x_1}(\tilde{g}_{i,2}(x_1, \beta_2 x_1 + \alpha_2),\ \tilde{g}_{j,2}(x_1, \beta_2 x_1 + \alpha_2))$$

over $F[\alpha_2, \beta_2, x_1]$. It is easy to see that $0 \neq \pi_i \sigma_{i,j} \in F[\alpha_2, \beta_2]$ and $\pi_i(b_2)\sigma_{i,j}(a_2, b_2) \neq 0$ implies that the above events are impossible. Now, $\deg(\pi_i) \leqslant d_i$ and $\deg(\sigma_{i,j}) \leqslant 2d_i d_j$ and therefore the probability that the above events occur for any $i \neq j$ is less than

$$\sum_{i=1}^{r} \frac{d_i}{\text{card}(R)} + \sum_{1 \leqslant i < j \leqslant r} \frac{2d_i d_j}{\text{card}(R)} < \frac{(d_1 + \cdots + d_r)^2}{\text{card}(R)} < \frac{d^2}{\text{card}(R)} < \frac{\varepsilon}{6}.$$

Notice that if P were division-free, this event would be the only one where failure could occur.

Fifth, we may not find good interpolation points in order to produce Q_0. If we try at most $(d+1)^4$ points, the probability that at least $(d+1)^2 = d^*$ points are good can be estimated like in the proof of [K7], Theorem 5.1. We shall repeat the argument here. An individual point was not picked earlier with probability $\geqslant 1 - (d+1)^4/\mathrm{card}(R) > 1 - \varepsilon/12$. $\bar{P}$ is not defined at an individual point substituted for x_1 with probability $< 2^{l+1}/\mathrm{card}(R) < \varepsilon/12$. Hence a suitable point can be found in a block of d^* points with probability greater than

$$1 - (\varepsilon^*)^{d^*} > 1 - \frac{\varepsilon^*}{d^*}, \qquad \varepsilon^* = \frac{\varepsilon}{6},$$

because $(1/\varepsilon^*)^{d^*-1} > 2^{d^*-1} \geqslant d^*$ for $\varepsilon^* < 1/2$. Now the probability that a good point occurs in all of the d^* blocks of points is greater than

$$\left(1 - \frac{\varepsilon^*}{d^*}\right)^{d^*} > 1 - \varepsilon^*,$$

and therefore failure happens with probability $< \varepsilon/6$. Sixth and last, for $F = \mathbf{Q}$ we may not recognize that we have good interpolating points, that for all $(d+1)^2$ points together with probability $< \varepsilon/6$. Summing up these failure probabilities completes the proof. □

We remark that our result in [K3] would allow to further reduce the problem on an algebraic RAM over F to univariate factorization. We also mention that the input parameter d can be probabilistically estimated in expected polynomial-time in $\deg(f)$ (Section 3). Furthermore, the algorithm could be formulated using quadratic lifting [K2] in step L. Then the length of Q could be asymptotically reduced to $O(d^2 l + M(d^2)\log(d))$. Finally we mention that the binary polynomial-time upper bound can be easily generalized to F being an algebraic extension of $\mathbf{Q}$.

We now formulate two corollaries to Theorem 6.1. The first refers to computing the sparse factorization of f and follows from Theorem 4.2.

COROLLARY 6.1. If in addition to the input parameters of the Factorization algorithm we are given $t > 0$, for $F = \mathbf{Q}$ or

$F = \mathbf{F}_q$ we can find polynomially many binary steps and random bit choices in

$$l, d, \log\left(\frac{1}{\varepsilon}\right), \text{el-size}(P), \text{cc-size}(f), \text{ and } t$$

sparse polynomials that with probability $> 1 - \varepsilon$ constitute all irreducible factors of f with no more than t monomials. □

Notice that the above running time is always polynomial independent whether the correct sparse factors were produced or whether other factors are dense. This makes this corollary superior to all previous work on sparse factorization. The second corollary deals with possibility nonuniform closure. Again, in a family of p-computable polynomials the degrees computation lengths are polynomially bounded [V2].

COROLLARY 6.2. Let F be a field of characteristic 0. Then any family of factors of a family of p-computable polynomials over F is p-computable. □

Notice that this corollary applies even to fields in which arithmetic is recursive but over which polynomial factorization is undecidable [FS]. It also shows that a polynomial degree bound is necessarily required. We note that $x^{2^d} - 1$ can be computed with $O(d)$ instructions but it is known that over the complex numbers there exist factors that require $\Omega(2^{d/2}/\sqrt{d})$ computation length [LS, Sc]. It would be nice to give such an example where the factors are irreducible polynomials over $\mathbf{Q}$.

We have implemented the Factorization algorithm [FI]. In order that len(Q) does not become too large, two practically important improvements to the Factorization algorithm as it is described above were made. First, the coefficients $\bar{f}_{j,m}$ are not computed a priori but as they are needed for each k in the lifting loop. This is accomplished by using the Polynomial Coefficients algorithm in the original verssion of [K7], which is based on Taylor series expansion. Second, the product $\prod_{i=1}^{r} g_{i,k}^{e_i}$ is also computed incrementally using the coefficients determined already for $k - 1$ of the same product.

7. CONCLUSION

Aside from the preceding paper [K7] currently two more of our papers deal with the subject of manipulating polynomials in straight-line representation. In the forthcoming paper [K10] we show how to replace the input parameter d in the Factorization algorithm by a degree bound for the individual factors. We also have implemented our algorithms in Lisp with an interface to Macsyma. The details of this first implementation together with practical improvements and our experience on test cases are reported in [FI].

The question arises as to what major unresolved problems in the subject of polynomial factorization remain. It is appropriate to distinguish between theoretical and practical issues. One theoretical question is to remove the necessity of random choices from any of the problems known to lie within probabilistic polynomial-time, say factorization of univariate polynomials over large finite fields. Another problem is to investigate the parallel complexity of polynomial factorization, say for the **NC** model [Co]. Kronecker's reduction from algebraic number coefficients [Kr, T, L], Berlekamp's factorization algorithm over small finite fields [B1], the author's deterministic Hilbert irreducibility theorem [K3, §7], and Weinberger's irreducibility test for $\mathbf{Q}[x]$ [We] all lead to **NC** solutions by simply applying known **NC** methods for linear algebra problems. It is open whether factoring in $\mathbf{Q}[x]$ or irreducibility testing in $\mathbf{F}_p[x]$, p large, or in $\mathbf{Q}[x, y]$ can be accomplished in **NC**. We remark that testing a rational dense multivariate polynomial for absolute irreducibility can be shown to be in **NC** [K6].

In connection with the Factorization algorithm presented here, we also mention an open question. Assume that a straight-line program computes a polynomial whose degree is exponential in the length of the program. Are its factors at least of polynomially bounded degree p-computable? A positive answer to this question would show that testing a polynomial for zero in a suitable decision-tree model is polynomial-time related to computing that polynomial (cf. [K10, §6, Problem 6]). In general the theory of straight-line manipulation of polynomials may be extendable in part to unbounded input degrees, but even for the elimination of divisions problem [S2] the answer is not known.

From a pragmatic point of view the main unresolved question is what role the polynomial-time polynomial factorization algorithms

should play in computer algebra systems, that is in actually used implementations. The "L^3" algorithm [LLL] has been considered impractical by even one of the inventors, but that was not meant to imply that this algorithm is useless for polynomial factorization. In fact, using L^3 to recover algebraic numbers from their modular images leads to a practically competitive factoring algorithm for polynomials over algebraic number fields [Le1]. We submit that careful implementations of different lattice reduction schemes together with the complex root approximation method [Sh2] might outperform the Berlekamp–Hensel algorithm on hard-to-factor polynomials. The first implementation of the straight-line factorization algorithm is reported in [FI]. There its practical merits have been demonstrated on very dense inputs such as symbolic determinants.

In summary, in this chapter we were able to contribute to Valiant's algebraic counterpart of the theory of **P** vs. **NP** in the positive, that is establish a polynomial upper bound for a major problem in computational algebra. In fact, it comes to us as a small surprise that p-computable polynomials are closed under factorization. And we have, finally, put to rest the problem of computing the sparse factorization of a multivariate polynomial.

ACKNOWLEDGMENTS

A discussion I had with Barry Trager a few years ago has helped me in developing Section 4. I also thank Joachim von zur Gathen and Gregory Imirzian for their valuable comments.

This material is based upon work supported by the National Science Foundation under Grant No. DCR-85-04391 and by an IBM Faculty Development Award. The results of Section 6 were originally announced in the paper "Uniform closure properties of p-computable functions" [K9].

NOTE

Since this chapter has been submitted, progress on several problems discussed can be reported. The sparse conversion problem in Section 4 has been solved more efficiently by Ben-Or and Tiwari [*Proc. 20th Annual ACM Symp. Theory Comput.* 301–309 (1988)], Zippel [*J. Symbolic Comput.* to appear (1990)], and by Lakshman Yagati and the author [*Proc. ISSAC 1988, Springer Lec. Notes Comput. Sci.* to appear (1989)]. John Canny and Barry Trager have made the author aware of a more effective version of the Hilbert Irreducibility Theorem 5.1, that essentially reduces the numerator of the probability bound to $d^{O(1)}$. Such a theorem also

follows from methods presented in [K6], Section 5. Finally, Barry Trager and the author [*Proc. 29th Annual Symp. Foundations Comput. Sci.* 296–305 (1988)] have shown that another implicit representation for multivariate polynomials, that of black box programs that merely allow to evaluate the polynomials at given input points, can be used as input and output representation for polynomial-time polynomial factorization.

REFERENCES

[BS] W. Baur and V. Strassen, "The complexity of partial derivatives," *Theor. Comp. Sci.* 22: 317–330 (1983).

[B1] E. R. Berlekamp, "Factoring polynomials over finite fields," *Bell Systems Tech. J.* 46: 1853–1859 (1967). Republished in revised form in E. R. Berlekamp, *Algebraic Coding Theory*, Chapter 6. McGraw-Hill, New York, 1968.

[B2] E. R. Berlekamp, "Factoring polynomials over large finite fields," *Math. Comp.* 24: 713–735 (1970).

[CG] A. L. Chistov and D. Yu. Grigoryev, "Polynomial-time factoring of multivariable polynomials over a global field," *LOMI preprint E-5-82*, Steklov Institute, Leningrad, 1982.

[Cl] B. G. Claybrook, "A new approach to the symbolic factorization of multivariate polynomials," *Artificial Intelligence* 7: 203–241 (1976).

[Cd] G. E. Collins, "Factoring univariate integral polynomials in polynomial average time," *Proc. EUROSAM '79, Springer Lec. Notes Comp. Sci.* 72: 317–329 (1979).

[Co] S. A. Cook, "A taxonomy of problems with fast parallel algorithms," *Inf. Control* 64: 2–22 (1985).

[FI] T. S. Freeman, G. Imirzian, E. Kaltofen and L. Yagati, "DAGWOOD: A system for manipulating polynomials given by straight-line programs," *ACM Trans. Math. Software* 14: 218–240 (1988).

[FS] A. Fröhlich and J. C. Shepherdson, "Effective procedures in field theory," *Phil. Trans. Roy. Soc., Ser. A* 248: 407–432 (1955/56).

[G1] J. von zur Gathen, "Parallel algorithms for algebraic problems," *SIAM J. Comp.* 13: 802–824 (1984).

[G2] J. von zur Gathen, "Irreducibility of multivariate polynomials," *J. Comp. System Sci.* 31: 225–264 (1985).

[G3] J. von zur Gathen, "Parallel arithmetic computation: A survey," *Proc. MFCS '86, Springer Lec. Notes Comp. Sci.* 233: 93–112.

[GK1] J. von zur Gathen and E. Kaltofen, "Factoring multivariate polynomials over finite fields," *Math. Comp.* 45: 251–261 (1985).

[GK2] J. von zur Gathen and E. Kaltofen, "Factoring sparse multivariate polynomials," *J. Comp. System Sci.* 31: 265–287 (1985).

[Ge] A. O. Gelfond, *Transcendental and Algebraic Numbers*. Dover, New York, 1960.

[He] J. Heintz, "A note on polynomials with symmetric galois group which are easy to compute," *Theor. Comp. Sci.* 47: 99–105 (1986).

[HS] J. Heintz and M. Sieveking, "Absolute primality of polynomials is decidable in random polynomial-time in the number of variables," *Proc. ICALP '81, Springer Lec. Notes Comp. Sci.* 115: 16–28 (1981).

[Hen] K. Hensel, *Theorie der algebraischen Zahlen*. Teubner, Leipzig, 1908.

[Hi] D. Hilbert, "Über die Irreduzibilität ganzer rationaler Funktionen mit ganzzahligen Koeffizienten," *J. reine angew. Math.* 110: 104–129 (1892).

[Hu] M.-D. A. Huang, "Riemann hypothesis and finding roots over finite fields," *Proc. 17th ACM Symp. Theory Comp.* 121–130 (1985).

[IM] O. H. Ibarra and S. Moran, "Probabilistic algorithms for deciding equivalence of straight-line programs," *J. ACM* 30: 217–228 (1983).

[Je] R. D. Jenks, "A primer: 11 keys to new SCRATCHPAD," *Proc. EUROSAM '84, Springer Lec. Notes Comp. Sci.* 174: 123–147 (1984).

[JK] D. E. Jordan, R. Y. Kain, and L. C. Clapp, "Symbolic factoring of polynomials in several variables," *Comm. ACM* 9: 638–643 (1966).

[K1] E. Kaltofen, "A polynomial reduction from multivariate to bivariate integral polynomial factorization," *Proc. 14th Annu. ACM Symp. Theory Comp.* 261–266 (1982).

[K2] E. Kaltofen, "Polynomial factorization," In *Computer Algebra*, 2nd ed. (B. Buchberger et al., eds.), pp. 95–113. Springer-Verlag, Vienna, 1982.

[K3] E. Kaltofen, "Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization," *SIAM J. Comp.* 41: 469–489 (1985).

[K4] E. Kaltofen, "Effective Hilbert irreducibility," *Inf. Control* 66: 123–137 (1985).

[K5] E. Kaltofen, "Computing with polynomials given by straight-line programs II; Sparse factorization," *Proc. 26th IEEE Symp. Foundations Comp. Sci.* 451–458 (1985).

[K6] E. Kaltofen, "Fast parallel absolute irreducibility testing," *J. Symbolic Comp.* 1: 57–67 (1985).

[K7] E. Kaltofen, "Greatest common divisors of polynomials given by straight-line programs," *J. ACM* 35/1: 231–264 (1988).

[K8] E. Kaltofen, "Uniform closure properties of p-computable functions," *Proc. 18th ACM Symp. Theory Comp.* 330–337 (1986).

[K9] E. Kaltofen, "Deterministic irreducibility testing of polynomials over large finite fields," *J. Symbolic Comp.* 3: 77–82 (1987).

[K10] E. Kaltofen, "Single-factor Hensel lifting and its application to the straight-line complexity of certain polynomials," *Proc. 19th Annu. ACM Symp. Theory Comp.* 443–452 (1987).

[K11] E. Kaltofen, D. R. Musser, and B. D. Saunders, "A generalized class of polynomials that are hard to factor," *SIAM J. Comp.* 12: 473–485 (1983).

[Kn] D. E. Knuth, *The Art of Programming*, Vol. 2, *Semi-Numerical Algorithms*, ed. 2. Addison-Wesley, Reading, MA, 1981.

[Kr] L. Kronecker, "Grundzüge einer arithmetischen Theorie der algebraischen Grössen," *J. reine angew. Math.* 92: 1–122 (1882).

[L] S. Landau, "Factoring polynomials over algebraic number fields," *SIAM J. Comp.* 14: 184–195 (1985).

[LM] S. Landau and G. L. Miller, "Solvability by radicals," *J. Comp. System Sci.* 30: 179–208 (1985).

[Le1] A. K. Lenstra, "Lattices and factorization of polynomials over algebraic number fields," *Proc. EUROCAM '82, Springer Lec. Notes Comp. Sci.* 144: 32–39 (1982).

[Le2] A. K. Lenstra, "Factoring multivariate integral polynomials," *Theor. Comp. Sci.* 34: 207–213 (1984).

[Le3] A. K. Lenstra, "Factoring multivariate polynomials over finite fields," *J. Comp. System. Sci.* 30: 235–248 (1985).

[LLL] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients," *Math. Ann.* 261: 515–534 (1982).

[LS] R. Lipton and L. Stockmeyer, "Evaluations of polynomials with superpreconditioning," *Proc. 8th ACM Symp. Theory Comp.* 174–180 (1976).

[M1] D. R. Musser, "Multivariate polynomial factorization," *J. ACM* 22: 291–308 (1975).

[M2] D. R. Musser, "On the efficiency of a polynomial irreducibility test," *J. ACM* 25: 271–282 (1978).

[N] I. Newton, *Arithmetica Universalis*, 2nd ed., London, 1728. In English. Reprinted in *The Mathematical Works of Isaac Newton*, Vol. 2 (D. T. Whiteside, ed.). Johnson Reprint Corp., New York, 1967.

[PS] M. S. Paterson and L. J. Stockmeyer, "On the number of nonscalar multiplications necessary to evaluate polynomials," *SIAM J. Comp.* 2: 60–66 (1973).

[R1] M. O. Rabin, "Probabilistic algorithms for testing primality," *J. Number Theory* 12: 128–138 (1980).

[R2] M. O. Rabin, "Probabilistic algorithms in finite fields," *SIAM J. Comp.* 9: 273–280 (1980).

[Sc] C. P. Schnorr, "Improved lower bounds on the number of multiplications/divisions which are necessary to evaluate polynomials," *Theor. Comp. Sci.* 7: 251–261 (1978).

[Sh] R. J. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p," *Math. Comp.* 44: 483–494 (1985).

[Sw] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *J. ACM* 27: 701–717 (1980).

[Sh1] A. Schönhage, "Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2," *Acta Inf.* 7: 395–398 (1977). (In German.)

[Sh2] A. Schönhage, "Factorization of univariate integer polynomials by diophantine approximation and an improved basis reduction algorithm," *Proc. ICALP '84, Springer Lec. Notes Comp. Sci.* 172: 436–447 (1984).

[SS] R. M. Solovay and V. Strassen, "A fast Monte-Carlo test for primality," *SIAM J. Comp.* 6: 84–85 (1977). Correction: 7: 118 (1978).

[S1] V. Strassen, "Berechnung und Programm I," *Acta Inf.* 1: 320–335 (1972). (In German.)

[S2] V. Strassen, "Vermeidung von Divisionen," *J. reine angew. Math.* 264: 182–202 (1973). (In German.)

[S3] V. Strassen, "Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten," *Numer. Math.* 20: 238–251 (1973). (In German.)

[S4] V. Strassen, "Polynomials with rational coefficients which are hard to compute," *SIAM J. Comp.* 3: 128–149 (1974).

[T] B. M. Trager, "Algebraic factoring and rational function integration," *Proc. 1976 ACM Symp. Symbolic Algebraic Comp.* 219–228 (1976).

[V1] L. Valiant, "The complexity of computing the permanent," *Theor. Comp. Sci.* 8: 189–201 (1979).

[V2] L. Valiant, "Reducibility by algebraic projections," *L'Enseignement math.* 28: 253–268 (1982).

[vW] B. L. van der Waerden, *Modern Algebra.* F. Ungar, New York, 1953.

[W] P. S. Wang, "An improved multivariate polynomial factorization algorithm," *Math. Comp.* 32: 1215–1231 (1978).

[We] P. J. Weinberger, "Finding the number of factors of a polynomial," *J. Algorithms* 5: 180–186 (1984).

[Y] D. Y. Y. Yun, "The Hensel lemma in algebraic manipulation," Ph.D. Thesis, M.I.T., 1974. Reprint: Garland Publ., New York, 1980.

[Z] H. Zassenhaus, "On Hensel factorization I," *J. Number Theory* 1: 291–311 (1969).

[Zi1] R. E. Zippel, "Probabilistic algorithms for sparse polynomials," *Proc. EUROSAM '79, Springer Lec. Notes Comp. Sci.* 72: 216–226 (1979).

[Zi2] R. E. Zippel, "Newton's iteration and the sparse Hensel algorithm," *Proc. '81 ACM Symp. Symbolic Algebraic Comp.* 68–72 (1981).

A RANDOMIZED DATA STRUCTURE FOR ORDERED SETS

Jon L. Bentley, F. Thomson Leighton, Margaret Lepley,

Donald F. Stanat, and J. Michael Steele

ABSTRACT

In this chapter, we consider a simple randomized data structure for representing ordered sets, and give a precise combinatorial analysis of the time required to perform various operations. In addition to a practical data structure, this work provides new and nontrivial probabilistic lower bounds and an instance of a practical problem whose randomized complexity is provably less than its deterministic complexity.

1. INTRODUCTION

In this chapter, we consider the problem of maintaining a set from a totally ordered domain under the operations Member, Insert,

Advances in Computing Research, Volume 5, pages 413–428.

ISBN: 0-89232-896-7

Delete, Predecessor, Successor, Maximum, and Minimum. The basic data structure that we use to represent such a set of size at most N is a sorted linked list implemented by the two arrays Link[0...N] and Value[1...N]. The value in Link[0] points to the first element in the list, Link[Link[0]] points to the next element, etc. We call such a data structure a *J-list* after Janko [J1, J2] who first studied the structure in a randomized setting. Among other things, we will show that Member, Insert, Predecessor, Successor, and Maximum can all be accomplished in $2\sqrt{N} - c$ expected steps where c is a small constant, and that this bound is optimal under a plausible model of constraints imposed by the data structure. We also show that Delete requires just $4\sqrt{N} - 2c$ expected steps and that Minimum requires just one step (Link[0] points to the minimum element). All of these bounds (except for Minimum) are dramatically better than the worst case bound of N steps.

Although quite simple, the J-list data structure is surprisingly efficient. In fact, it is superior to all of those described by Knuth [K] for certain applications. The salient attributes of such applications are as follows:

- Space is important. This structure uses only one extra word of storage per element, while binary search trees use at least two extra words, and various hashing schemes use varying amounts of extra storage. However, the storage for this structure must be available in a single contiguous block.
- The "orderedness" operations of Successor, Predecessor, Minimum, and Maximum are frequent; these are not possible in most hashing schemes.
- Insertions and deletions are frequent. If the data structure changes rarely, binary search in a sorted array is very efficient.
- Program simplicity is important. Each operation on this structure requires only about a dozen lines of code, while some operations on balanced binary search trees require over 100 lines of code.
- Run time is important for problems of medium size (where medium means that N is between, say, 100 and 10,000). If N is below that range, simple sequential strategies are probably efficient enough. If N is above that range, then the logarithmic search time of binary search will be necessary for many applications. When N is in the medium range, though, the low constant factors of this structure will make it competitive with binary search trees.

Of course, the simple linked list is one of the most basic and well-known data structures, and has arisen in countless contexts. Most relevant to this chapter is the prior work of Janko [J1, J2] who studied randomized algorithms for sorting using linked lists and obtained an $O(N^{3/2})$ bound on the expected time needed to sort N items.

The remainder of the chapter is divided into sections as follows. In Section 2, we define the problem more precisely and observe that the worst-case complexity of performing a Member Search is linear in N. For the most part, we concentrate our efforts on the analysis of a simple algorithm for Member Search. This is because the algorithm for Member Search can be easily transformed into an efficient algorithm for each of the other operations. In Section 3, we describe the J-list structure and explain its relationship to a simple Guess-Decrement game. We also describe an optimal randomized algorithm for Member Search and show how it can be extended to form efficient algorithms for Insert, Delete, and the other operations. Section 4 considers some natural extensions of the basic model and contains some additional probabilistic analysis.

2. THE PROBLEM AND ITS DETERMINISTIC COMPLEXITY

The J-list is implemented in contiguous storage by the two arrays Link[0 . . . N] and Value[1 . . . N]. The pointer Link[0] points to the first element of the list, Value[Link[0]]. The next element can be found in Value[Link[Link[0]]], and so forth. The end of the list is denoted by an element whose Link field contains -1. Furthermore, we will insist that the array is *dense*: Value[1 . . . N] must contain N elements of the represented set. The sortedness of the linked list implies that if Link[I] is not -1, then Value[I] $\leqslant$ Value[Link[I]]. We will often refer to Value[I] and Link[I] together as node I. Figure 1 illustrates the array representation of the sorted linked list $\langle 2.6, 3.1, 4.1, 5.3, 5.8, 5.9, 9.7\rangle$.

It is clear that performing a Member Search in such an array requires accessing at most N elements of the array (either by following Link fields through the list or simply by iterating through Value fields of the array). We will now show that in the worst case, this

Figure 1. An array representation of the sorted linked list $\langle 2.6, 3.1, 4.1, 5.3, 5.8, 5.9, 9.7\rangle$.

I	0	1	2	3	4	5	6	7
Value[I]		3.1	4.1	5.9	2.6	5.3	5.8	9.7
Link[I]	4	2	5	7	1	6	3	−1

much time is necessary to decide whether a given element is in the list. We will assume that a (deterministic) search algorithm is composed of operations of the following types, each with unit cost.

1. Determine the index of the node at the head of the list (by accessing Link[0]). There is one operation of this type.
2. Determine the successor of node I for $1 \leqslant I \leqslant N$ (by accessing Link[I]). There are N operations of this type.
3. Determine the Value of node I, for $1 \leqslant I \leqslant N$ (by accessing Value[I]). There are N operations of this type.

(Note that if operations of type 2 have no cost, then binary search can be used to solve the problem in logarithmic time.)

Our model assumes that a protagonist specifies a sequence of the above operations while an adversary ensures that N operations will be required. We will assume that the adversary knows the value of the key the protagonist seeks, which we will call V, and that other key values may be assigned arbitrarily by the adversary. We will describe a strategy that enables the adversary to delay returning V until the protagonist has specified a sequence of N operations. Without loss of generality, we will assume that whenever one of Value[I] or Link[I] is asked, both of Value[I] and Link[I] are provided at a cost of a single step ($1 \leqslant I \leqslant N$). The value of V will be the maximum element in the list. There are two cases depending on whether or not the protagonist asks the type 1 question.

Case 1: The type 1 question is not asked.

The adversary always answers questions so that the protagonist has queried a contiguous subset of the ordered list. In particular, assume that the protagonist asks about node I where I has not yet been queried. (Remember that Value[I] and Link[I] are always provided together, at a total cost of one.) If $I =$ Link[J] where J is the node at the head of the continuous subset of previously queried nodes, then the adversary assigns the largest value yet given (but less than V, of course) to Value[I] and assigns an as yet unqueried node to Link[I]. If $I \neq$ Link[J], then the adversary assigns the smallest value yet given to Value[I] and sets Link[I] $= K$ where K is the smallest node in the contiguous subset of previously queried nodes. In either case, the set of queried nodes continues to form a contiguous subset of ordered list, with all values less than V.

The argument continues in this fashion until $N - 1$ nodes have been queried. At this point, the remaining unqueried node is $I = \text{Link}[J]$ where J is the largest queried node. In order to resolve Member Search for V, the protagonist must still ask the value of node I, making for a total of N queries overall.

Case 2: The type 1 question is asked.

In this case, we will show that $N - 1$ nodes must be queried, making for a total of N steps. The argument proceeds as before until the protagonist asks the type 1 question. In response, the adversary reveals that $\text{Link}[0] = K$ where K is the smallest node in the contiguous subset of previously queried nodes. From this point on, the adversary will answer questions so that the queried nodes form at most two contiguous subsets of the ordered list, one of them beginning with Link[0]. The subset beginning with Link[0] will always have values smaller than the other contiguous subset, and all values will be less than V. The details of the adversary's responses are similar to before until a total of $N - 2$ node queries have been made. At this point the protagonist must still query the value of node I where $I = \text{Link}[J]$ and J is the node with the largest value seen so far. Hence $N - 1$ queries need to be made, accounting for N operations overall.

3. RANDOMIZED ALGORITHMS

In what follows, we focus on algorithms that allow probabilistic access to nodes in addition to deterministic and Link access. Although the worst case performance of such randomized algorithms is no different than that for deterministic algorithms, we will find that the average case performance is much better.

The section is divided into subsections as follows. In Section 3.1, we define a class of simple randomized algorithms for Member Search. We model the performance of algorithms in this class with a Guess-Decrement game in Section 3.2. In Section 3.3, we use the game model to show that the expected running time for the optimal Member Search Algorithm is $2\sqrt{N} - c$, where c is a small constant. We extend the algorithm for Member Search to other operations in Section 3.4.

3.1. A Class of Randomized Algorithms for Member Search

By combining probabilistic access with access by predecessor link, a wide range of algorithms can be considered. For example, the following pseudo-Pascal program searches for the element E, using the order of operations specified by the array Step. When Step$[J]$ is zero, a random sample occurs. Otherwise the program follows the next link in the list. Note that when a random sample is chosen, the position in the list is updated only if the random position is closer to (but not at or beyond) the location of E. This strategy ensures that the updated position in the list never worsens and that when E is eventually found, its predecessor will also have been found (since E will have been reached via a link). (This particular code assumes that Value[0] is $-\infty$ and Value$[-1]$ is ∞.)

```
P := 0
J := 0
do until exit
  J := J + 1
  if Step[J] = 0 then do
    R := Random(1, N)
    if Value[R] < E and Value[R] > Value[P] then P := R
  else do
    if Value[Link[P]] = E then exit (*E is at Link[P]*)
    if Value[Link[P]] > E then exit (*E is not in the list*)
    if Value[Link[P]] < E then P := Link[P]
```

For any specified Step array, the expected performance of the associated algorithm will depend on the value of E being searched. For example, if E is less than or equal to the smallest item in the list, then the algorithm will terminate on or before the first Link access. For the time being, we will focus on the more interesting case when E is bigger than the largest item in the list. This is, in fact, the worst case for any step array in terms of expected running time, and is representative of the case when E has a random rank. For expediency, we will defer the proof of these assertions to Section 4, where we consider search values with an arbitrary index. We also consider more sophisticated algorithms in Section 4, including procedures that decide whether to step forward or move randomly based on whether or not previous random moves were successful.

3.2. Modeling Algorithms as Strategies

Before proceeding to construct an optimal algorithm from the class described in Section 3.1, it is useful to associate algorithms in the class with strategies for a simple probabilistic "Guessing-Decrement" game. The *G-D Game* involves two integers, i and N. The value of N remains fixed throughout the game, and the value of i is originally N. The goal of the player is to reduce the value of i to zero in the minimum expected number of steps. A step consists of performing one of the following two operations:

- D (for Decrement): If $i > 0$, then replace i by $i - 1$.
- G (for Guess): Choose j to be an integer uniform from $1 \ldots N$ and replace i by j if $j < i$. The value of i is unknown to the player, except at the beginning of the game when $i = N$, and at the end when he is notified that i has reached zero.

The value of i represents the distance from the current element in the linked list—denoted by P in the above algorithm—to the end of the list. The value N is the number of items in the list. We start with $i = N$ because we assume that we are searching for the largest element in the list. In the general case, we would start with i equal to the rank of E, if known.

Each Guess corresponds to a random access in the above code, whereas each Decrement corresponds to a link access. A strategy or sequence of operations will be denoted by a character string σ composed of G's and D's to be performed in order from left to right. A sequence of G's and D's corresponds naturally to a Step array. A sequence is said to be *complete* if it contains at least N D's. Note that operations written after the first N D's are superfluous and need not appear. For convenience, however, we will often end complete sequences with D^N.

A complete sequence will always reach $i = 0$ and terminate the game after some number of steps t. The expected termination point for a complete sequence σ is denoted by

$$\mathrm{E}(\sigma) = \sum_{j=1}^{\infty} j \Pr[t = j] = \sum_{j=0}^{\infty} \mathrm{Q}_j(\sigma)$$

where $\mathrm{Q}_j(\sigma) = \Pr[t > j]$. The object is to minimize $\mathrm{E}(\sigma)$ over all complete sequences σ. We denote the minimum by $S(N)$.

Note that $E(\sigma)$ also denotes the expected running time of the Member Search algorithm for the corresponding Step array. Hence, determining the optimal σ is equivalent to determining the optimal algorithm from the class described in Section 3.1. For simplicity, we will use the G-D notation henceforth.

3.3. An Optimal Strategy

The task of finding an optimal strategy for the G-D game will proceed in two steps. The first step consists of finding the best strategy from among those of the form G^kD^N for some k. The second, and more difficult, step consists of showing that there is an optimal strategy having this form.

We will start by analyzing strategies of the form G^kD^N. Within this restricted class, it is easy to determine the best values of k and the minimum of $E(G^kD^N)$.

THEOREM 1. Let $k(N)$ be the value of k that minimizes $E(G^kD^N)$. Then $k(N) = \sqrt{N} - 1 - 1/(24\sqrt{N}) + O(1/N)$.

Proof. From the definition,

$$E(G^kD^N) = \sum_{j=0}^{k+N} Q_j(G^kD^N).$$

Since the first k operations are Guesses the game cannot end there, so $Q_j(G^kD^N) = 1$ for $j = 0, \ldots, k$. The probability of not terminating during the first d Decrements is $(N-d)^kN^{-k}$, since all the Guesses must be larger than d in order not to terminate. Therefore

$$\begin{aligned} E(G^kD^N) &= k + 1 + \sum_{d=1}^{N-1} (N-d)^kN^{-k} \\ &= k + 1 + N^{-k} \sum_{d=1}^{N-1} d^k \\ &= k + 1 + N/(k+1) - 1/2 + k/(12N) + O(k^2/N^2). \end{aligned}$$

The minimum occurs when

$$1 - N/(k+1)^2 + 1/(12N) + O(k/N^2) = 0$$

and thus when

$$k = \sqrt{N} - 1 - 1/(24\sqrt{N}) + O(1/N). \qquad \square$$

Theorem 1 provides an upper bound of $S(N) \leqslant 2\sqrt{N} - 1/2 + 1/(12\sqrt{N}) + O(1/N)$ expected operations for the G-D game. Of course, the optimal value of $k = \sqrt{N} - 1 - 1/(24\sqrt{N}) + O(1/N)$ may have to be rounded to a neighboring integer, so we should conclude only that $S(N) \leqslant 2\sqrt{N} - c$ where c is a small constant that tends to $1/2$ as N grows large. In what follows, we will show that this bound is tight by proving that there is an optimal strategy of the form $G^k D^N$. We commence with some definitions and lemmas.

When a sequence ω is not complete, the value of i after the operations in ω have been performed may remain undetermined. Instead of knowing the exact value of i we define a probability vector

$$P_j(\omega) = \Pr[i > j \text{ after executing the sequence } \omega].$$

We can see from the definition that $P_j(\omega) \geqslant P_{j+1}(\omega)$. Moreover the vector can be computed for any sequence ω.

LEMMA 1. *For any sequence* $\omega = G^{a_1} D^{b_1} \ldots G^{a_s} D^{b_s}$, *with* $b = b_1 + \cdots + b_s$ *and* $a = a_1 + \cdots + a_s$

$$P_j(\omega) = \begin{cases} (N-j-b)^{a_1}(N-j-b+b_1)^{a_2} & \\ \quad \cdots (N-j-b_s)^{a_s} N^{-a} & \text{for } j < N-b \\ 0 & \text{for } j \geqslant N-b. \end{cases}$$

Proof. During each block of Guesses, G^{a_m}, the value i must remain above j plus the number of D's that are still to be performed, $b_m + \cdots + b_s$. The probability that all the Guesses in the block are between $j + b_m + \cdots + b_s + 1$ and N is $(N - j - b_m - \cdots - b_s)N^{-a_m}$. $\square$

These probabilities are important in determining the optimal strategy. The following lemma states one of the most useful properties of this vector.

LEMMA 2. *For any sequence* ω, $P_j(\omega D)/P_0(\omega D) \leqslant P_j(\omega)/P_0(\omega)$ *for* $0 \leqslant j \leqslant N$.

Proof. Each ratio is a product of terms of the form

$$[(N - j - b_i - \cdots - b_s)/(N - b_i - \cdots - b_s)]^{a_i}.$$

The sequence ωD has one more D in the last block than ω, so b_s increases by one in ωD. When $P_0(\omega D) > 0$, a comparison of these terms, letting $K = N - b_i - \cdots - b_s$, reveals that

$$[(K - j - 1)/(K - 1)]^{a_i} < [(K - j)/K]^{a_i}$$

and thus that $P_j(\omega D)/P_0(\omega D) \leqslant P_j(\omega)/P_0(\omega)$. If $P_0(\omega D) = 0$, we define the ratio to be zero and the inequality still holds. □

Remember that our goal is to prove that the optimal strategy has the form $G^k D^N$. To do this, we next analyze the effect of minor variations in strategy on the expected number of operations. Then we will show that if a small variation improves the strategy, then a larger change could mean even more improvement. The two sequences which we will compare first are $\sigma = \omega D G^k D^N$ and $\sigma^* = \omega G^k D^N$. The only difference between σ and σ^* is the position of the block G^k. The following lemma gives a method for comparing these two strings.

LEMMA 3. $E(\omega D G^k D^N) \leqslant E(\omega G^k D^N)$ *if and only if* $V_k(\omega) \leqslant 1$, *where*

$$V_k(\omega) = \begin{cases} P_1(\omega)/P_0(\omega) + \sum_{d=1}^{N-1} P_d(\omega)/P_0(\omega) \\ \quad \times [(N - d + 1)^k - (N - d)^k] k^{-1} N^{-k} & \text{if } P_0(\omega) > 0 \\ 0 & \text{if } P_0(\omega) = 0. \end{cases}$$

Proof. Let σ and σ^* be defined as above. We would like to know when $E(\sigma) - E(\sigma^*) \leqslant 0$. The following values for $Q_m(\sigma)$ and $Q_m(\sigma^*)$ can be easily verified.

$$Q_m(\sigma) = \begin{cases} Q_m(\sigma^*) & \text{if } m \leqslant |\omega| \\ P_1(\omega) & \text{if } |\omega| + 1 \leqslant m \\ & \quad \leqslant |\omega| + k + 1 \\ P_{m-|\omega|-k}(\omega) & \\ \quad \times (N - m + |\omega| + k + 1)^k N^{-k} & \text{if } m > |\omega| + k + 1 \end{cases}$$

$$Q_m(\sigma^*) = \begin{cases} Q_m(\sigma) & \text{if } m \leqslant |\omega| \\ P_0(\omega) & \text{if } |\omega| + 1 \leqslant m \\ & \quad \leqslant |\omega| + k \\ P_{m-|\omega|-k}(\omega) & \\ \quad \times (N - m + |\omega| + k)^k N^{-k} & \text{if } m > |\omega| + k. \end{cases}$$

Thus $E(\sigma) - E(\sigma^*) = \sum_{m=0}^{\infty} [Q_m(\sigma) - Q_m(\sigma^*)] \leqslant 0$ is equivalent to

$$(k+1)P_1(\omega) - kP_0(\omega) + \sum_{d=1}^{N-1} P_d(\omega)[(N-d+1)^k - (N-d)^k]N^{-k} - P_1(\omega) \leqslant 0.$$

Rearranging terms slightly gives

$$P_1(\omega) + \sum_{d=1}^{N-1} P_d(\omega)[(N-d+1)^k - (N-d)^k]N^{-k}k^{-1} \leqslant P_0(\omega).$$

□

Combining Lemmas 2 and 3 enables us to extend a minor variation of the string into a more radical change. Specifically, if the last block of G's is more efficient when it is moved to the right one place, then it is best to remove the block of Guesses altogether.

LEMMA 4. *For all* ω, *if* $E(\omega DG^k D^N) \leqslant E(\omega G^k D^N)$, *then* $E(\omega D^N) \leqslant E(\omega DG^k D^N)$.

Proof. If $E(\omega D^j G^k D^N) \leqslant E(\omega D^{j-1} G^k D^N)$ for some $j \geqslant 1$, then $V_k(\omega D^{j-1}) \leqslant 1$ by Lemma 3. By Lemma 2 and the definition of $V_k(\omega)$, we know that $V_k(\omega D^j) \leqslant V_k(\omega D^{j-1})$, and thus that $V_k(\omega D^j) \leqslant 1$. Thus, we can conclude by Lemma 3 that $E(\omega D^{j+1} G^k D^N) \leqslant E(\omega D^j G^k D^N)$. The proof of the lemma is completed by applying this process inductively. □

It is now a simple matter to prove our main result.

THEOREM 2. *For every starting sequence* ω, *there exists an integer* $r \geqslant 0$ *such that* $E(\omega G^r D^N) \leqslant E(\omega\sigma)$ *for every completed sequence* $\omega\sigma$.

Proof. Let σ denote the shortest (in length) sequence for which $\omega\sigma$ is complete and $E(\omega\sigma) = \min_\gamma E(\omega\gamma)$. If σ is of the form $G^r D^N$,

then we are done. Otherwise $\sigma = \omega^* DG^k D^j$, where G^k is the last block of Guesses and $j > 0$. By the optimality of σ, we know that $E(\omega\omega^* DG^k D^j) \leqslant E(\omega\omega^* G^k D^{j+1})$. Thus by Lemma 4, we know that $E(\omega\omega^* D^{j+1}) \leqslant E(\omega\omega^* DG^k D^j)$ which contradicts the minimality of σ. □

Thus the best way to finish any initial sequence is by a block of Guesses followed by Decrements. By letting ω be the empty string, we find that the optimal strategy for the game is $G^k D^N$. Recalling Theorem 1, we find that the optimal value is near $\sqrt{N} - 1 - 1/(24\sqrt{N}) + O(1/N)$ and thus $S(N) = 2\sqrt{N} - c$ where c is a constant that tends to $1/2$ as N gets large. Hence the expected number of steps for Member Search is at most $2\sqrt{N}$. As a consequence, it is not difficult to show that this is within one or two steps of optimal whenever we are searching for an item that is bigger than the median. Searches for items less than the maximum are discussed more thoroughly in Section 4.

3.4. Algorithms for the Other Operations

Thus far we have considered only the problem of searching the linked list to determine if it contains a given element. It is easy to perform many other set operations on this structure. The following list summarizes those operations, and describes their costs in terms of the number of Value elements accessed.

- **Member**: The previous sections studied the problem of searching to determine whether a given element is a member of the set represented by the linked list. *Cost*: $2\sqrt{N}$.
- **Insert**: A new element can be inserted in the list by using Member Search. *Cost*: $2\sqrt{N}$.
- **Delete**: The first step in deleting an element is to find that element by a search algorithm, and then modify the Link field of its predecessor to point to its successor. This takes $2\sqrt{N}$ references to the Value array. The next step must patch the "hole" created in the dense array by moving the last element of the array to the vacant position. Searching for the last element requires $2\sqrt{N}$ references. *Cost*: $4\sqrt{N}$.
- **Predecessor**: The element immediately preceding a given element can be found by a simple modification to the Member Search algorithm. *Cost*: $2\sqrt{N}$.

- **Succcessor**: The element immediately succeeding a given element can be found by a simple modification to the Member Search algorithm. *Cost*: $2\sqrt{N}$.
- **Minimum**: The minimum element in the set is pointed to by Link[0]. *Cost*: 1.
- **Maximum**: The maximum element in the set can be found by searching for infinity. *Cost*: $2\sqrt{N}$.

Each of the above operations is straightforward to implement given the Member Search algorithm and basic techniques for dealing with data structures for searching (described, for example, by Knuth [K]). Furthermore, the simplicity of the algorithm implies that the constant factors in the running time of the program will be relatively small. The only deviation that a programmer should make from the Member Search algorithm deals with the random number generation: since some random number generators are very slow, it might be preferable to use some other approach to sample the k elements.

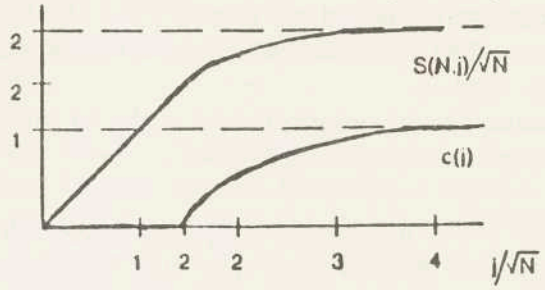
4. EXTENSIONS AND REMARKS

At first glance, it might appear that our proof technique depends on the fact that Guesses do not yield zero. This is not the case. When a Guess returns an integer uniform on $[0 \ldots N]$, the optimal strategy differs from the above strategy by only $O(1/\sqrt{N})$ Guesses.

Our analysis of the G-D game was for the worst case task of searching for the final element in the list. We will now consider searches for a random element. The value i can be interpreted as the number of links between our present position in the list and the position of the element we are searching for. When searching for the last element we start at $i = N$. Suppose a Member search seeks an element whose position is unknown and randomly distributed. We should then start at an unknown, random i, or equivalently start at $i = N$ and do one Guess to randomize i before counting operations. By Theorem 2 the optimum strategy is then $G^{r-1}D^N$ as opposed to G^rD^N when searching for the last element.

Sometimes it is useful to find the jth element given j. When j is small it is easy to follow links and when $j = N$ we can use the G-D strategy. Between these two extremes, the strategies used thus far are not necessarily valid since the value of the jth element is

Figure 2. Graph of $c(j)$ and $S(N, j)/\sqrt{N}$ where $S(N, j)$ is the optimal expected number of operations when searching for the jth element.



unknown. But if we know the value as well as the position of the element for which we are searching, then we can apply the above techniques to find an optimal G-D search time. Searching for an element at position j means starting the G-D game at $i = j$. This is equivalent to starting at $i = N$ and doing $N - j$ Decrements. By Theorem 2 the best way to continue from this point is $G^k D^N$ for some k. Thus it is only necessary to compute the optimal number of Guesses, $k = r(j)$. By modifying Theorem 1 it can be shown that $r(j) = c(j)\sqrt{N}$ where

$$c(j) = \begin{cases} 0 & \text{if } j \leqslant \sqrt{2N} \\ (j-1)^{-1}\sqrt{N}\{e^{(j-1)c(j)\sqrt{N}} & \\ \quad \times [1 - c(j)^2] - 1\} - O(1/\sqrt{N}) & \text{if } \sqrt{2N} < j \leqslant O(\sqrt{N}) \\ 1 - O(1/\sqrt{N}) & \text{if } j \gg \Omega(\sqrt{N}). \end{cases}$$

It is easy to determine $c(j)$ numerically and a graph of the function (e.g., see Figure 2) shows that (once nonzero) $c(j)$ approaches 1 exponentially fast.

The G-D game can be modified in other ways. One particularly interesting modification allows the player to use the information about when a random sample is successful (i.e., closer to the target). For example, suppose the value i is contained in a black box that is connected to a light that turns on every time i is decreased. At first glance, it appears as though such information could be quite useful in planning when to stop Guessing and start Decrementing. For instance, if the light flashed on for a series of early Guesses, then the player might be led to believe that the early Guesses were very good and thus that i had become very small. Hence, the player might

think it wise to start Decrementing early. This is *not* the case, however, since if all the Guesses are required to be distinct, then it can be shown that the light does not add any useful information at all. It is worth remarking that the likelihood of two Guesses being identical is small and thus the constraint that all the Guesses be different has a negligible effect on the final result.

Such a counterintuitive result requires some justification. First notice that when all the Guesses are distinct, the sequence of Guesses is just a permutation of a subset R of $\{1, \ldots, N\}$. Every sequence of Guesses produces a unique light sequence, β, but one light sequence can be produced by many different Guess sequences. In particular:

LEMMA 5. *For every light sequence β of length k there exists an integer m, such that for any set of Guesses $R \subseteq \{1, \ldots, N\}$ of size k there are exactly m permutations of R that have light sequence β.*

Proof. Let $K = \{1, \ldots, k\}$ and set m to be the number of permutations of K that produces the light sequence β. Now consider any other subset R of length k. There is a $1 - 1$ order-preserving mapping between K and R, so there is also a $1 - 1$ mapping between the permutations of the two sets that preserves light sequences. This means that there are also m permutations of R that fit β. □

We can now prove

THEOREM 3. *When all the Guesses in the G-D game are distinct, then the light sequence adds no extra information about the value of i after a sequence of Guesses.*

Proof. It is sufficient to show that the probability $\Pr[i = j \mid \beta]$ that $i = j$ after k guesses given a light sequence β, is the same as the probability $\Pr[i = j]$ that $i = j$ after k guesses (with no knowledge of the light sequence). From the definitions and Lemma 5,

$$\begin{aligned}
\Pr[i = j \mid \beta] &= (\#\text{ of sequences of Guesses that fit } \beta \text{ for which } i = j)/ \\
&\quad (\#\text{ of sequences of Guesses that fit } \beta) \\
&= (\#\text{ of } R \text{ for which } i = j)m/(\#\text{ of } R)m \\
&= (\#\text{ of } R \text{ for which } i = j)k!/(\#\text{ of } R)k! \\
&= (\#\text{ of sequences of Guesses for which } i = j)/ \\
&\quad (\#\text{ of sequences of Guesses}) \\
&= \Pr[i = j].
\end{aligned}$$

□

The G-D game might also be played with two different operations O_1 and O_2. It would be interesting to know what properties of O_1 and O_2 give an optimal sequence of the form $O_1^k O_2^m$, for some k and m. In addition to operators that act directly on i, we might also consider comparison operations that compare i to a given input n, and answer the question, "$i \leqslant n$?" If the compare operation $i \leqslant \sqrt{2N}$ is added to G-D, then the optimal strategy is $O(N^{1/4})$ Guesses followed by a Compare, repeated until $i \leqslant \sqrt{2N}$, ending with Decrements. This strategy uses $\sqrt{2N} + O(N^{1/4})$ expected steps.

ACKNOWLEDGMENTS

We would like to thank Gary Miller, Ron Rivest, Jim Saxe, and Mike Sipser for helpful discussions.

This research was supported in part by ONR Contract N00014-76-C-0370, NSF Grant MCS-78-07736, and an NSF Presidential Young Investigator Award with matching funds from Xerox and IBM. Parts of this chapter were presented at the 19th and 20th Annual Allerton Conferences on Communication, Control and Computing [BSS, LL].

REFERENCES

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

[BSS] J. L. Bentley, D. F. Stanat, and J. M. Steele, "Analysis of a randomized data structure for representing ordered sets," *Proc. 19th Annu. Allerton Conference on Communication, Control, and Computing* 364–372 (1981).

[J1] W. Janko, "A list insertion sort for keys with arbitrary key distribution," *ACM Transact. Math. Software* 2: 143–153 (1976).

[J2] W. Janko, "An insertion sort for uniformly distributed keys based on stopping theory," *Int. Comp. Symp.* April 1977, pp. 373–379. North-Holland Publishing, Amsterdam, 1977.

[K] D. E. Knuth, *The Art of Computer Programming*. Vol. 3: *Sorting and Searching*. Addison-Wesley, Reading, MA, 1973.

[LL] T. Leighton and M. Lepley, "Probabilistic searching in sorted linked lists," *Proc. 20th Annu. Allerton Conference on Communication, Control and Computing* 500–506 (1982).

ON COMPLETENESS AND SOUNDNESS IN INTERACTIVE PROOF SYSTEMS

Martin Furer, Oded Goldreich, Yishay Mansour,

Michael Sipser, and Stathis Zachos

ABSTRACT

An interactive proof system with *Perfect Completeness* (resp. *Perfect Soundness*) for a language L is an interactive proof (for L) in which for every $x \in L$ (resp. $x \notin L$) the verifier *always* accepts (resp. *always* rejects). We show that any language having an interactive proof system has one (of the Arthur–Merlin type) with perfect completeness. On the other hand, only languages in NP have interactive proofs with perfect soundness.

1. INTRODUCTION

The two basic notions regarding a proof system are *completeness* and *soundness*. Completeness means that the proof system is

Advances in Computing Research, Volume 5, pages 429–442.

ISBN: 0-89232-896-7

powerful enough to generate "proofs" for all the valid statements (in some class). Soundness means that any statement that can be proved is valid (i.e., no "proofs" exist for false statements). Two computational tasks related to a proof system are generating a proof and verifying the validity of a proof. This naturally suggests the notions of a *prover* (a party able of generating proofs) and a *verifier* (a party capable of validating proofs). Typically, the verifier's task is easier than the prover's task. In order to focus on the complexity of the verification task it is convenient to assume that the prover has unlimited power.

For many years *NP* was considered *the formulation* of "*whatever can be efficiently verified.*" This stemmed from the association of deterministic polynomial-time computation with efficient computation. The growing acceptability of probabilistic polynomial-time computations as reflecting efficient computations is the basis of more recent formalizations of "whatever can be efficiently verified." In these formalizations, due to Goldwasser, Micali, and Rackoff [GMR] and Babai [B], and shown to be equivalent by Goldwasser and Sipser [GS], the (polynomial time) verifier is allowed to toss coins and arbitrarily interact with the prover; furthermore he can accept or reject based on overwhelming statistical evidence. Ruling by overwhelming statistical evidence means relaxing the completeness and soundness conditions so that any valid statement can be proved with a very high probability while any false statement has only negligible probability to be proved. For a definition of *interactive proof systems* we refer the reader to Goldwasser and Sipser's article [GS] in this volume.

We denote by *IP* the class of languages for which there exists an interactive proof system. Clearly, $NP \subseteq IP \subseteq PSPACE$. It is believed that the class *NP* is *strictly* contained in *IP*. Evidence for this may perhaps be derived from the fact that, relative to some oracle, interactive proofs are even not contained in the polynomial-time hierarchy, i.e., $\exists A$ s.t. $IP^A - PH^A \neq \varnothing$ (see [AGH]). It is also interesting to note that natural languages such as Graph Non-isomorphism and Matrix Group Nonmembership, which are not known to be in *NP*, were shown to be in *IP* by [GMW] and [B], respectively.

Considering an interactive proof system, it seems that in some sense the prover is "responsible" for the completeness condition, while the verifier is "responsible" for the soundness condition. If this intuition is correct, and the prover has unrestricted power, why

this intuition is correct, and the prover has unrestricted power, why should the completeness condition be relaxed? Namely, can one modify the interactive proof such that the prover never fails in demonstrating the validity of true statements, while maintaining soundness. By *perfect completeness* we mean that the prover *never* fails to prove the membership of inputs that are indeed in the language, while *perfect soundness* means that the verifier *never* accepts inputs that are not in the language.

Perfect completeness and perfect soundness are not only theoretically interesting, but are also of practical importance. This is the case, since probabilistic completeness and soundness are defined with respect to ideal (unbiased) coin tosses and may not hold when using pseudorandom sequences (even in the sense of Blum and Micali [BM] and Yao [Y]). On the other hand perfect completeness and soundness are independent of the quality of the verifier coin tosses.

Our main result is that *Interactive Proofs with Perfect Completeness* are as **powerful** *as Interactive Proofs*. Now, what about interactive proofs with perfect soundness? Unfortunately, we show that they are only as powerful as *NP*.

The proof of the main result is in fact a transformation that given an interactive proof for a language L yields an Arthur–Merlin interactive proof with perfect completeness for L. This transformation preserves the number of interactions of the original interactive proof. An alternative proof that uses different ideas, and in particular a protocol for "random selection," appears in [GMS]. An alternative characterization of complexity classes defined by bounded Arthur-Merlin games, and their perfect completeness, appears in [ZF].

2. MODEL AND DEFINITIONS

We state and prove our main result for the Arthur–Merlin games introduced by Babai [B]. Using the result of [GS] our main result applies also to the interactive proof systems of [GMR]. In this section we provide a precise definition of Arthur–Merlin games and auxiliary terminology, in order to facilitate the presentation of our result.

Since we are interested in the complexity theoretic aspects of proof systems, we may assume that the prover (Merlin) uses an

optimal strategy and, therefore, with no loss of generality, is deterministic. In the following definition we assume that in all interactions of Arthur and Merlin, on inputs of the same length, the same number of messages are exchanged and that all these messages are of the same length. Clearly, this condition is immaterial and is placed only in order to facilitate the analysis.

DEFINITION 1 (ARTHUR–MERLIN GAMES). An *Arthur–Merlin game* is a pair of interactive programs **A** and **M** and a function ρ such that

1. On common input x, exactly $2q(|x|)$ messages of length $m(|x|)$ each are exchanged, where q and m are fixed polynomials and $|x|$ denotes the length of x.
2. Arthur (**A**) goes first, and an iteration $1 \leqslant i \leqslant q(|x|)$ chooses at random a string r_i of length $m(|x|)$, with uniform probability distribution.
3. Merlin's reply in the ith iteration, denoted y_i, is a function of all the previous choices of Arthur and the common input x. More formally, $y_i = \mathbf{M}(x, r_1 \cdots r_i)$. In other words, **M** is the strategy of Merlin.
4. For every program $\mathbf{M}'$, a *conversation* between **A** and $\mathbf{M}'$ on input x is a string $r_1 y_1 \cdots r_{q(|x|)} y_{q(|x|)}$, where for every $1 \leqslant i \leqslant q(|x|)\, y_i = \mathbf{M}'(x, r_1 \cdots r_i)$. We denote by $CONV_x^{\mathbf{M}'}$ the set of all conversations between **A** and $\mathbf{M}'$ on input x. Note that $|CONV_x^{\mathbf{M}'}| = 2^{q(|x|)m(|x|)}$.
5. The function ρ is a polynomial-time computable predicate. This predicate maps the input x and a conversation $r_1 y_1 \cdots r_{q(|x|)} y_{q(|x|)}$ to a Boolean value, called *the value of the conversation* (i.e., $\rho(x, r_1 y_1 \cdots r_{q(|x|)} y_{q(|x|)}) \in \{accept, reject\}$). The function ρ is called the *value-of-the-game function.*

Notation. Let **A** and $\mathbf{M}'$ be programs and ρ be a function as above. Then $ACC_x^{\rho, \mathbf{M}'}$ denotes the set $\{r_1 \cdots r_{q(|x|)} \mid \exists y_1 \cdots y_{q(|x|)}$ s.t. $r_1 y_1 \cdots r_{q(|x|)} y_{q(|x|)} \in CONV_x^{\mathbf{M}'}$ and $\rho(r_1 y_1 \cdots r_{q(|x|)} y_{q(|x|)}) = accept\}$.

Intuitively, $ACC_x^{\rho, \mathbf{M}'}$ is the set of all the random choices leading **A** to accept x, when interacting with $\mathbf{M}'$. Note that $ACC_x^{\rho, \mathbf{M}'}$ depends only on Merlin ($\mathbf{M}'$) and the function ρ, since we assume that Arthur follows the protocol. The ratio $|ACC_x^{\rho, \mathbf{M}'}|/|CONV_x^{\mathbf{M}'}|$ is the probability that Arthur accepts x when interacting with $\mathbf{M}'$.

DEFINITION 2 (ARTHUR–MERLIN PROOF SYSTEMS). An *Arthur–Merlin proof system for language* L is an Arthur–Merlin game satisfying the following two conditions:

1. There exists a strategy for Merlin, $\mathbf{M}$, such that for all $x \in L$, $|ACC_x^{\rho,\mathbf{M}}|/|CONV_x^{\mathbf{M}}| \geq \frac{2}{3}$. (This condition is hereafter referred to as *probabilistic-completeness*.)
2. For every $\mathbf{M}'$ and for any $x \notin L$, $|ACC_x^{\rho,\mathbf{M}'}|/|CONV_x^{\mathbf{M}'}| \leq \frac{1}{3}$. (This condition is hereafter referred to as *probabilistic-soundness*.)

An equivalent definition is obtained by replacing 1/3 by $2^{-p(|x|)}$ and 2/3 by $1 - 2^{-p(|x|)}$, where $p(\cdot)$ is an arbitrary polynomial satisfying $p(n) > 1$ (for $\forall n > 1$).

DEFINITION 3 (PERFECT COMPLETENESS). An Arthur–Merlin proof system with *perfect-completeness* for a language L is an Arthur–Merlin proof system for L satisfying

$$\forall x \in L \qquad |ACC_x^{\rho,\mathbf{M}}| = |CONV_x^{\mathbf{M}}|.$$

Perfect-completeness, of an Arthur–Merlin proof system, means that an honest Merlin always succeeds in convincing Arthur to accept inputs in the language.

DEFINITION 4 (PERFECT SOUNDNESS). An Arthur–Merlin proof system with *perfect-soundness* for a language L is an Arthur–Merlin proof system for L satisfying

$$\forall \mathbf{M}' \qquad \forall x \notin L \qquad ACC_x^{\rho,\mathbf{M}'} = \varnothing.$$

Perfect-soundness, of an Arthur–Merlin proof system, means that no matter what Merlin does Arthur never accepts an input not in language.

3. ARTHUR–MERLIN PROOF SYSTEMS WITH PERFECT COMPLETENESS

In this section we transform an Arthur–Merlin proof system to an Arthur–Merlin proof system with perfect completeness. This transformation preserves the number of interactions in the original

Arthur–Merlin proof. The underlying technique is taken from Lautemann's proof that BPP is in the polynomial-time hierarchy [L]. (Lautemann's proof that BPP is in the polynomial-time hierarchy simplifies the original proof of Sipser [S].) The idea is to show that this technique works also for Arthur–Merlin proof systems. We think that this idea seems strange at first glance, trivial in second thought, but in fact is quite surprising and important.

Lautemann's technique is commonly presented as a method of expressing a "random" quantifier by a universal and an existential quantifier. Suppose we are dealing with a subset, W, of $\{0,1\}^k$ and that this subset has cardinality either $\geqslant (1-\varepsilon)\cdot 2^k$ or $\leqslant \varepsilon \cdot 2^k$. The statement "most $r \in \{0,1\}^k$ are in W" can be substituted by the statement "$\exists s^{(1)}, s^{(2)}, \ldots, s^{(k)} \in \{0,1\}^k$ such that $\forall r \in \{0,1\}^k$ there $\exists i$ $(1 \leqslant i \leqslant k)$ such that $s^{(i)} \oplus r \in W$," where $s \oplus r$ is the bit-by-bit XOR of the strings s and r. These strings are said to "*cover*" W. The statement "most $r \in \{0,1\}^k$ are not in W" can be substituted by the statement "$\forall s^{(1)}, s^{(2)}, \ldots, s^{(k)} \in \{0,1\}^k \exists r \in \{0,1\}^k \forall i$ $(1 \leqslant i \leqslant k)$ $s^{(i)} \oplus r \notin W$."

Zachos showed that the above "simulation" can be used to switch quantifiers in a successive manner (for survey see [Z, Sch]). Zachos and Fuerer [ZF] then used this idea to show that bounded Arthur–Merlin proofs equal bounded Arthur–Merlin proofs with perfect completeness, by expressing the former proofs as a fixed quantifier sequence and applying a "switching lemma" iteratively. Each such iteration is thus a straightforward application of the "simulation technique," and blows-up the size of the Arthur–Merlin game by an unbounded amount. Thus, this idea does not extend to unbounded Arthur–Merlin proofs.

For our transformation it is necessary to extend the simulation technique to settings in which the witness set W is not predetermined. In fact, in Arthur–Merlin games the set of random choices leading Arthur to accept is not defined, unless Merlin is specified. This fact is disturbing in the case that the input is not in the language and one has to guarantee that *no matter how Merlin acts* he cannot fool Arthur (except for with low probability).

3.1. An Overview of the Protocol

Without loss of generality, we assume that the error probability in the original Arthur–Merlin game is sufficiently small [i.e., $\varepsilon(|x|) < 1/[3q(|x|)m(|x|)]$]. The transformed Arthur–Merlin game will consist of $k = q(|x|)m(|x|)$ original games played concurrently

with related coin tosses, and Arthur will accept iff he accepts in one of these games. More specifically, Merlin starts the game by selecting carefully k strings, $s^{(1)}, s^{(2)}, \ldots, s^{(k)} \in \{0,1\}^k$, and sending them to Arthur. These strings are selected to "cover" $ACC_x^{\rho,\mathbf{M}}$ in the case that x is in the language. Arthur and Merlin now start to play k copies of the original game. In round j, Arthur sends only one m-bit string r_j and his move in the ith game is defined as the bit-by-bit XOR of r_j and the jth segment in $s^{(i)}$ (i.e., Arthur's jth move in the ith copy is $r_j^{(i)} = r_j \oplus s_j^{(i)}$, where $s_j^{(i)}$ is the jth m-bit block in $s^{(i)}$). Merlin answers by k strings so that the ith string equals the answer the original Merlin would have given in the ith copy [i.e., the ith m-bit block in Merlin's jth message equals $\mathbf{M}(x, r_1^{(i)} r_2^{(i)} \cdots r_j^{(i)})$, where $\mathbf{M}$ is the original Merlin]. Clearly, the perfect completeness condition is satisfied. It is less easy to see that probabilistic soundness is satisfied as well. Note that a cheating Merlin may select his answers for one copy of the game depending on his prospects in the other copies, and in particular the structure of $ACC_x^{\rho,\mathbf{M}}$ is irrelevant. Our argument, instead, consists of two claims: (1) the probability of winning the transformed game is bounded by the sum of the probabilities of winning each copy; and (2) the probability of winning a particular copy is bounded by the probability of winning the original game. (Trying to incorporate both claims in one counting argument leads to difficulties that are not encountered in Lautemann's original proof.)

3.2. The Protocol

We denote the original Arthur by $\hat{\mathbf{A}}$, the original Merlin by $\hat{\mathbf{M}}$, and the original value-of-the-game function by $\hat{\rho}$. Let ε be the error probability, i.e., for $x \in L$ the Prob($\hat{\mathbf{A}}$ accepts) $> 1 - \varepsilon(|x|)$, and for $x \notin L$ the Prob($\hat{\mathbf{A}}$ accepts) $< \varepsilon(|x|)$. On input of size n, $q(n)$ iterations are performed, at each iteration Arthur sends a message of length $m(n)$. When clear from the text we use ε, q, m for $\varepsilon(n), q(n), m(n)$, respectively. Let $k = qm$. Without loss of generality we assume that $\varepsilon < (1/3k)$. This can be achieved by performing sufficiently many copies of the original Arthur–Merlin game in parallel, and ruling by the majority (see [B], [GS], and [BHZ]).

3.2.1. *Program for an Honest Merlin*

Merlin's program consists of two stages. First, Merlin computes k "sampling points" that are favorable to him, and sends them to

Arthur. The second stage is a simulation of k (related) copies of the original Arthur–Merlin game.

Preprocessing stage: Let ACC be the set of random choices leading Arthur to accept in the original $\hat{\mathbf{A}}\hat{\mathbf{M}}$ game on input $x \in L$ (i.e., ACC is a shorthand for $ACC_x^{\hat{\rho},\mathbf{M}}$). Merlin selects k strings $s^{(1)}, s^{(2)}, \ldots, s^{(k)} \in \{0,1\}^k$ so that for every $r \in \{0,1\}^k$ there exists an i such that $s^{(i)} \oplus r \in ACC$. The preprocessing is said to have *failed*, if no such set of $s^{(i)}$'s exist. If the preprocessing does not fail then Merlin sends the $s^{(i)}$'s to Arthur. For sake of simplicity, we let Merlin send k (arbitrary) strings (of length k-bit each) in case the preprocessing fails.

Simulation stage: Merlin plays concurrently k copies of the original game and computes Arthur's responses by XORing them with segments of the $s^{(i)}$'s. Each $s^{(i)}$ is partitioned into q segments, of m bits each, corresponding to the q iteration of the original game. Namely, $s^{(i)} = s_1^{(i)} s_2^{(i)} \cdots s_q^{(i)}$, where $s_j^{(i)} \in \{0,1\}^m$. Formally, at each iteration j $[1 \leqslant j \leqslant q(n)]$, Merlin performs

Receive r_j
For $i = 1$ to k do begin
    $r_j^{(i)} \leftarrow s_j^{(i)} \oplus r_j$
    $y_j^{(i)} \leftarrow \mathbf{M}(x, r_1^{(i)} \cdots r_j^{(i)})$
End
Send $y_j^{(1)}, y_j^{(2)}, \ldots, y_j^{(k)}$.

3.2.2. *Arthur's Program*

Arthur's program is identical to the original program of Arthur. Formally, for each iteration j $[1 \leqslant j \leqslant q(n)]$ Arthur performs

Choose r_j at random in $\{0,1\}^m$.
Send r_j
Receive $y_j^{(1)}, y_j^{(2)}, \ldots, y_j^{(k)}$.

3.2.3. *The Value of a Conversation*

Let $r_j^{(i)} = r_j \oplus s_j^{(i)}, y_j = y_j^{(1)} y_j^{(2)} \cdots y_j^{(k)}$, and $\bar{s} = s^{(1)} s^{(2)} \cdots s^{(k)}$. We denote by

$$\rho_i(x, \bar{s} r_1 y_1 \cdots r_q y_q) = \hat{\rho}(x, r_1^{(i)} y_1^{(i)} \cdots r_q^{(i)} y_q^{(i)})$$

the value of the ith game. The predicate ρ_i maps a conversation to 1 if and only if the conversation induced on the ith copy of the original game is an accepting one.

The value of a conversation is determined by the following polynomial-time predicate:

$$\rho(x, \bar{s}r_1 y_1 \cdots r_q y_q) = \bigvee_{i=1}^{k} \rho_i(x, \bar{s}r_1 y_1 \cdots r_q y_q).$$

3.3. The Perfect-Completeness of the Protocol

We show that if the input x is in L, then an honest Merlin always convinces Arthur. The argument is almost identical to the one in Lautemann (since ACC is fixed!), and is given here for the sake of self-containment.

LEMMA 1. If $x \in L$ then the preprocessing does not fail.

Proof. We have to show that if $|ACC| = (1 - \varepsilon) \cdot 2^k$ and $\varepsilon \leqslant (1/3k)$ then there exists a sequence, $\bar{s} = s^{(1)}, s^{(2)}, \ldots, s^{(k)}$ ($s^{(i)} \in \{0,1\}^k$), such that for every string $r \in \{0,1\}^k$ at least one of the $r \oplus s^{(i)}$ is in ACC. Furthermore, we will show that the statement holds for most sequences $\bar{s}$. We call a sequence $\bar{s} = s^{(1)}, s^{(2)}, \ldots, s^{(k)}$ *good* if for every $r \in \{0,1\}^k$ there exists an i ($1 \leqslant i \leqslant k$) such that $r \oplus s^{(i)} \in ACC$. We consider the probability that a randomly selected sequence $\bar{s}$ is not good.

$$\begin{aligned}
Prob(\bar{s} \text{ is not good}) &= Prob(\exists r \forall i\, r \oplus s^{(i)} \notin ACC) \\
&\leqslant \sum_{r \in \{0,1\}^k} Prob(\forall i\, r \oplus s^{(i)} \notin ACC) \\
&= 2^k \cdot Prob(\forall i\, s^{(i)} \notin ACC) \\
&= 2^k \cdot \varepsilon^k \\
&< \left(\frac{2}{3k}\right)^k < 2^{-k}.
\end{aligned}$$

The Lemma follows. □

LEMMA 2. If $x \in L$ then Arthur always accepts.

Proof. By Lemma 1, Merlin can find $s^{(i)}$'s so that (when Merlin follows his program!) any sequence of choices made by Arthur leads

to acceptance in at least one of the copies of the original game. The Lemma follows. □

3.4. The Probabilistic Soundness of the Protocol

We now show that for every input x not in L, no matter what Merlin does, the probability that he convinces Arthur is less than 1/3. We consider the probabilities that Merlin $\mathbf{M}'$ leads Arthur to accept in the ith copy of the original game. We first bound by ε the probability that $\mathbf{M}'$ leads Arthur to accept in the ith copy of the original game (see Lemma 3). Hence, the probability that Merlin cheats Arthur is bounded $k \cdot \varepsilon$ (Lemma 4).

Let $\mathbf{M}'$ be any arbitrary program for Merlin. Recall that $ACC_x^{\hat{\rho},\mathbf{M}'}$ denotes the set of random choices leading Arthur ($\hat{\mathbf{A}}$) to accept in the original game (with game value function $\hat{\rho}$). We denote the set of random choices leading Arthur to accept in the ith game of the transformed game by $ACC_x^{\rho_i,\mathbf{M}'}$. Namely, $r_1 r_2 \cdots r_q \in ACC_x^{\rho_i,\mathbf{M}'}$ iff $\rho_i[x, r_1\mathbf{M}'(x, r_1)\cdots r_q\mathbf{M}'(x, r_1 \cdots r_q)] = 1$. Note that both $ACC_x^{\rho_i,\mathbf{M}'}$ and $ACC_x^{\hat{\rho},\mathbf{M}}$ are subsets of $\{0,1\}^k$.

LEMMA 3. Suppose that $x \notin L$. Then for every Merlin $\mathbf{M}'$ and for every i $(1 \leqslant i \leqslant k)$ $|ACC_x^{\rho_i,\mathbf{M}'}| \leqslant \varepsilon \cdot 2^k$.

Proof. The idea of the proof is that a Merlin that does well on a particular copy of the original game can be easily transformed into a Merlin that does (at least) as well in the original game. The transformed Merlin (which plays the original game) simulates the actions of the Merlin that plays k games concurrently, using the real game as the ith copy. A detailed proof follows.

Assume, contrary to the statement of the lemma, that there exists an $\mathbf{M}'$ and an i, such that $|ACC_x^{\rho_i,\mathbf{M}'}| > \varepsilon \cdot 2^k$. We reach a contradiction by constructing a Merlin $\mathbf{M}''$, which does as well in the original game. First, $\mathbf{M}''$ runs $\mathbf{M}'$ on input x to get the k sample points $s^{(1)}, s^{(2)}, \ldots, s^{(k)}$ and saves $s^{(i)}$. Let $r_1, r_2, \ldots, r_j$ be the first j messages that $\mathbf{M}''$ has received. To compute the jth message, $\mathbf{M}''$ computes $r'_t = r_t \oplus s_t^{(i)}$ (for $1 \leqslant t \leqslant j$) and runs $\mathbf{M}'$ on input x and $r'_1 r'_2 \cdots r'_j$ [i.e., $\mathbf{M}''(x, r_1 \cdots r_j)$ is the ith m-bit block of $\mathbf{M}'(x, r'_1 \cdots r'_j)$]. We now claim that

CLAIM. $r \in ACC_x^{\rho_i,\mathbf{M}'}$ if and only if $r \oplus s^{(i)} \in ACC_x^{\hat{\rho},\mathbf{M}''}$.

Proof. Suppose that $r_1 \cdots r_q \in ACC^{\rho_i, \mathbf{M}'}$. Then $\rho_i(x, \bar{s} r_1 y_1 \cdots r_q y_q) = 1$, where $\bar{s} = s^{(1)} \cdots s^{(k)} = \mathbf{M}'(x)$ and $y_j = \mathbf{M}'(x, r_1 \cdots r_j)$ (for $\forall j$). It follows that $\hat{\rho}(x, r_1^{(i)} y_1^{(i)} \cdots r_q^{(i)} y_q^{(i)}) = 1$, where $y_j^{(i)}$ is the ith m-bit block in y_j, $s_j^{(i)}$ the jth m-bit block in $s^{(i)}$, and $r_j^{(i)} = r_j \oplus s_j^{(i)}$. Note that $y_j^{(i)} = \mathbf{M}''(x, r_1^{(i)} \cdots r_j^{(i)})$. Thus, $r_1^{(i)} \cdots r_q^{(i)} \in ACC_x^{\hat{\rho}, \mathbf{M}''}$. Noting that $r_1^{(i)} \cdots r_q^{(i)} = r \oplus s^{(i)}$ one direction follows. The proof of the second direction is similar and the claim follows. □

By the above Claim, $|ACC_x^{\hat{\rho}, \mathbf{M}''}| = |ACC_x^{\rho_i, \mathbf{M}'}| > \varepsilon \cdot 2^k$, which contradicts the hypothesis that the original game has error probability $\leqslant \varepsilon$. The lemma follows. □

REMARK. A statement analogue to Lemma 3 is trivial in Lautemann's setting.

LEMMA 4. Suppose that $x \notin L$. Then for every Merlin $\mathbf{M}'$ the probability that Arthur accepts is $\leqslant k \cdot \varepsilon$.

Proof. Clearly, for every Merlin $\mathbf{M}'$,

$$|ACC_x^{\rho, \mathbf{M}'}| = \left| \bigcup_{i=1}^{k} ACC_x^{\rho_i, \mathbf{M}'} \right| \leqslant \sum_{i=1}^{k} |ACC_x^{\rho_i, \mathbf{M}'}|.$$

Using Lemma 3, the statement follows. □

3.5. Main Result

Using the equivalence of interactive proofs and Arthur Merlin Proofs [GS], and combining Lemmas 2 and 4 we get

MAIN THEOREM (THEOREM 5). *If a language L has an interactive proof system [with $q(\cdot)$ iterations] then L has an (Arthur–Merlin) interactive proof system with perfect completeness [and $q(\cdot) + 1$ iterations].* □

4. INTERACTIVE PROOF SYSTEMS WITH PERFECT SOUNDNESS

In the previous section, we showed that interactive proofs can be modified so that the verifier always accepts valid statements. What happens if we require that the verifier never accepts false statements? In this case we show that the set of languages recognized equals NP.

The reader should note that the transformation of Goldwasser and Sipser [GS] does not preserve perfect completeness. Thus it is not clear that proving the above statement with respect to Arthur–Merlin games yields the same result with respect to general interactive proofs. The difficulty can be resolved by modifying the transformation of [GS], using the approximate lower bound protocol of [GMS] (which has the perfect completeness property). We prefer to give a direct proof.

The difference between interactive proofs and Arthur–Merlin games is that in interactive proofs the verifier's ith message α_i is a function of the input x, his random coin tosses r, and the previous messages of the prover [i.e., $\alpha_i = V(x, r, y_1 \cdots y_{i-1})$]. After the last (say qth) iteration, the verifier decides whether to accept or reject by evaluating the polynomial-time predicate $\rho(x, r, y_1 \cdots y_q) \in \{accept, reject\}$.

THEOREM 6. *If a language L has an interactive proof with perfect soundness then $L \in NP$.*

Proof. Assume that for a language L, there exists an interactive proof with perfect soundness. Since the verifier is limited to probabilistic polynomial time, then for any input $x \in L$ there is a conversation that convinces him, and is of polynomial length. The NP machine guesses this conversation, and checks that it is indeed a legitimate one and that is leads the verifier to accept. Namely, the machine guesses a random tape r and a conversation $\alpha_1 y_1 \cdots \alpha_q y_q$, and checks that $\alpha_i = V(x, r, y_1 \cdots y_{i-1})$ (for every i) and that $\rho(x, r, y_1 \cdots y_q) = accept$. If $x \in L$ then, by the probabilistic completeness condition, there exist (many) accepting conversations. If $x \notin L$ then, by the perfect-soundness condition, there is no such conversation, and any guess of the machine will fail. □

5. CONCLUDING REMARKS

Assuming the existence of secure encryption functions (in the sense of [GM]) and using the results of [GMW], one can easily demonstrate the existence of zero-knowledge interactive proofs with perfect completeness for every language in IP. Given $L \in IP$, first present an interactive proof with perfect completeness for L, and next apply

the techniques in [GMW] observing that they preserve perfect completeness. However, it is not clear whether every language having a perfect (resp. almost perfect) zero-knowledge interactive proof (see [F] for definition) has a perfect (resp. almost perfect) zero-knowledge interactive proof with perfect completeness. Weaker statement can nevertheless be proven:

1. Every language having an interactive proof that is almost perfect zero-knowledge *with respect to the specified verifier* has an interactive proof with perfect completeness that is almost perfect zero-knowledge *with respect to the specified verifier* (again see [F] for definition).
2. Every language having an interactive proof which is almost perfect zero-knowledge and remains so under *parallel composition* (see [O] for definition) has an almost perfect zero-knowledge proof with perfect completeness.

The key observation in proving both statements is that almost all sequences $\bar{s}$ can serve as sampling points (see proof of Lemma 1), and thus having the prover randomly select and send a *good* $\bar{s}$ does not yield any knowledge. (In the simulation we use a randomly selected $\bar{s}$, which is most likely but not necessarily good.)

Babai [B] showed that any Arthur–Merlin game with a fixed number of interactions can be simulated by a game with two interactions. A similar proof applies to the hierarchy of interactive proofs with perfect completeness.

Goldwasser and Sipser [GS] showed that the power of interactive proofs is not decreased when restricting the verifier to use only "public coins." We have showed that the power of interactive proofs is not decreased when further restricting the system to have perfect completeness. *How else can interactive proofs be restricted without decreasing their power?*

ACKNOWLEDGMENTS

We are grateful to the anonymous referee for his useful comments.

Oded Goldreich was partially supported by the Fund for Basic Research Administered by the Israeli Academy of Sciences and Humanities.

Stathis Zachos was partially supported by PSC-CUNY grant.

REFERENCES

[AGH] W. Aiello, S. Goldwasser, and J. Hastad, "On the power of interaction," *Proc. 27th FOCS* 368–379 (1986).

[B] L. Babai, "Trading group theory for randomness," *Proc. 17th STOC* 421–429 (1985).

[BM] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudorandom bits," *SIAM J. Computing* 13: 850–864 (1984).

[BHZ] R. Boppana, J. Hastad, and S. Zachos, "Does co-NP have short interactive proofs?," *IPL* 25: 127–132 (1987).

[F] L. Fortnow, "The complexity of perfect zero-knowledge," this volume.

[GMS] O. Goldreich, Y. Mansour, and M. Sipser, "Interactive proof systems: Provers that never fail and random selection," *Proc. 28th FOCS* (in press).

[GMW] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but the validity of the assertion and the mathodology of cryptographic protocol design," *Proc. 27th FOCS* 174–187 (1986).

[GM] S. Goldwasser and S. Micali, "Probabilistic encryption," *JCSS* 28(2): 270–299 (1984).

[GMR] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *Proc. 17th STOC* 291–304 (1985).

[GMS] O. Goldreich, Y. Mansour, and M. Sipser, "Interactive proof systems: Prover that never fail and random selection," *Proc. of the 28th IEEE Symp. FOCS* 449–461 (1987).

[GS] S. Goldwasser and M. Sipser, "Private coins versus public coins in interactive proof systems," this volume.

[L] C. Lautemann, "BPP and the polynomial-time hierarchy," *IPL* 14: 215–217 (1983).

[O] Y. Oren, "On the cunning power of cheating verifiers: Some observations about zero-knowledge proofs," M.Sc. thesis, in preparation.

[Sch] U. Schoening, "Probabilistic complexity classes and lowness," *Proc. 2nd Structure in Complexity Theory Conf., IEEE* 2–8 (1987).

[S] M. Sipser, "A complexity theoretic approach to randomness," *Proc. 15th STOC* 330–335 (1983).

[Z] S. Zachos, "Probabilistic quantifiers, adversaries, and complexity classes," *Proc. 1st Structure in Complexity Theory Conf.*, LNCS 223, pp. 383–400. Springer-Verlag, 1986.

[ZF] S. Zachos and M. Fuerer, "Probabilistic quantifiers vs. distrustful adversaries," unpublished manuscript, August 1985.

[Y] A. C. Yao, "Theory and applications of trapdoor functions," *Proc. of the 23rd IEEE Symp. FOCS* 80–91 (1982).

RANDOMIZATION IN BYZANTINE AGREEMENT

Benny Chor and Cynthia Dwork

ABSTRACT

This chapter surveys recent development and results in the area of randomized protocols for Byzantine Agreement.

1. INTRODUCTION

The Byzantine agreement problem, first posed by Pease, Shostak, and Lamport in 1980 [PSL80], has received more attention from the theoretical computer science community than any other problem in distributed computing. In this problem a collection of physically separated processors, some unknown subset of which may be faulty, must cooperatively determine a value that depends on pieces of information held privately by the individual processors, and

Advances in Computing Research, Volume 5, pages 443–497.

ISBN: 0-89232-896-7

all nonfaulty processors must agree on the resulting value. The difficulty is to achieve this agreement despite disruptive behavior by the faulty processors.

Byzantine agreement is viewed as the quintessential paradigm for distributed computing because of its wide applicability to real-life problems such as maintaining consistency of a distributed data base when updates are performed, ensuring mutual exclusion of processes competing for a critical resource, or allowing components in a redundant system to settle on a value, given slightly different readings from different sensors.

The randomized solutions to the Byzantine agreement problem discussed in this survey never sacrifice correctness: there is no execution in which any two correct processors disagree on the final value, and the required relationship between the inputs to the correct processors and the final decision value is never violated. In this respect the randomized protocols do not differ from deterministic protocols.

In discussing a deterministic protocol, the *cost* usually refers to the cost according to some measure, such as time or space, of the worst case execution of the given protocol. It is convenient to view the faulty processors as being selected and controlled by an adversary. If we think of the inputs as chosen by the adversary then cost is the worst case cost over all possible adversary strategies. (An adversary strategy includes the assignment of initial values, choice of faulty processors, and nature of their behaviour.) The same is true in a randomized algorithm, in which the processors may flip coins. However, in a randomized algorithm the coins are not considered to be under the control of, or even predictable by, the adversary. Given the protocol and the (possibly randomized) strategy $\mathcal{A}$ employed by the adversary, there is a well-defined probability space. One can now assign probabilities to events such as "the protocol has terminated after k steps," and calculate the expectation of the running time or any other cost measure of the protocol, under the specific strategy $\mathcal{A}$. The expected cost of the protocol is defined as the supremum, taken over all possible adversary strategies $\mathcal{A}$, of the expected cost of running the protocol with adversary strategy $\mathcal{A}$.

If, for a given problem $\mathcal{P}$, there exists a randomized protocol solving $\mathcal{P}$ whose expected cost is strictly less than the lower bound on the cost of solving $\mathcal{P}$ by a deterministic protocol, then "randomization provably helps" in solving problem $\mathcal{P}$. In this sense, randomization provably helps in solving the Byzantine agreement problem.

We begin our survey with a description of the model of computation, the types of failures studied, and the nature and power of the adversary controlling these failures. The subtleties of randomized protocols for Byzantine agreement are so challenging and of such a new nature that one cannot meaningfully discuss the recent developments without first conceptualizing (rather than merely listing) the failure models and the increasingly powerful type of adversaries being studied. We address these issues in Section 2 in some detail. We hope that such a coherent point of view will be useful to the community, especially in reducing the occurrence of such disturbing ambiguities as have plagued past work on this subject.

In order to help the reader appreciate what is gained by randomization, we briefly review some of the limitations of deterministic protocols for Byzantine agreement (see [F83] for a more complete survey of these and related results). We will then show how randomization provably helps in solving Byzantine agreement, by discussing several randomized protocols for this problem and comparing their expected costs to the corresponding lower bounds for deterministic protocols.

All the randomized protocols discussed require "coins" visible to certain (possibly large) subsets of the processors. The protocols differ in how these coins are used and in how the private random sources of the individual participants are combined to obtain the more widely visible coins. These two issues—how the coins are used, and how they are generated—are discussed in Sections 3 and 4, respectively.

We will also discuss some limitations on randomized protocols, mentioning the few known results in this area (Section 5). Open questions are discussed in Section 6. Finally, we include two appendices. Appendix I contains a list of randomized agreement protocols with succinct descriptions of the salient features of each, such as the resiliency, the expected running time, and the communication costs. Appendix II contains brief descriptions of cryptographic techniques used in some of the protocols. This will free us from stressing such details in the remainder of the text.

2. MODELS OF COMPUTATION, ADVERSARIES, AND COSTS

Byzantine agreement is an interesting problem only in the presence of (actual or potential) processor or message system failures. It is convenient to view faults as caused by an intelligent adversary.

After mentioning the basic components of a distributed system (Section 2.1), we discuss various assumptions commonly made about the adversary (Section 2.2). Of course, there is no way of actually controlling undesirable events in a distributed system. Thus, the assumptions simply specify the conditions under which a protocol is guaranteed to be correct. The goal in choosing the right assumptions is to cleanly model actual failure behavior. Section 2.3 discusses some cost measures by which the various protocols may be compared.

2.1. Basics of a Distributed System

A *distributed system* is a collection of physically separated processors that communicate by sending messages to one another. Some of the processors may be faulty. We say the system is *synchronous* if processing proceeds in synchronous rounds of message exchange. That is, we may imagine there is a global clock such that at each "tick" of the global clock all processors may send messages; moreover, messages sent between correct processors at one tick will be received before the next tick. A system that is not synchronous according to this definition will be called *asynchronous*. If both correct processors and the messages they send may suffer arbitrary delays then we say the system is *completely asynchronous*.

We think of the communication system as a complete network, with the vertices corresponding to the processors and the edges corresponding to communication wires. For the purposes of this survey each processor is considered to "know" the identity of the processor on the other end of each wire. We also assume the communication network is completely reliable, in that messages are never lost and messages sent by correct processors are not corrupted. In this case lost or corrupted messages are modeled by faulty behavior of either the sender or the intended receiver. Although Byzantine agreement has been studied in the context of unreliable communication links, we restrict our attention to the completely reliable case.

The processors themselves are (possibly infinite) state machines with special *input* and *output* registers for communication with the outside world and other processors. The protocols described in this survey require only finite control machines, such as interactive Turing machines, while the lower bounds and impossibility results hold even for infinite control machines. Each processor must have a source of randomness from which it can obtain unbiased and

unpredictable random bits as needed. We model this as either an unbiased coin, with values 0 and 1, or an infinite tape containing a truly random sequence of 0s and 1s.

Sometimes it is convenient to think of the processors as possessing private clocks. Viewed in this way, all private clocks of correct processors in a completely synchronous system are perfectly synchronized and run at exactly the same rate. Although Byzantine agreement has been studied in certain partially synchronous cases falling between the completely synchronous and the completely asynchronous models, most work on randomized protocols has involved only the two extremes.

A *protocol* is a collection of programs, one per processor. We assume the processors have been programmed before execution of the protocol begins.

2.2. The Adversary

There are five principal types of restrictions on the behavior of the adversary. Given a particular system, the user should decide which type of adversary best models system fault behavior and then use protocols resilient to this adversary.

The first parameter of the adversary is the upper bound t on the number of processors that can be corrupted by the adversary. There are several variants on the period of time during which this upper bound should hold. Certain protocols tolerate no more than t faults during the whole lifetime of the system. Other protocols are organized as a series of phases during each of which up to t processors can exhibit faulty behavior. Some protocols are more liberal, tolerating changes from one round to the next in the identity of the faulty processors, provided no more than t processors are faulty in any single round. We will be particularly interested in cases where t is a constant fraction of the number of participants, n.

The second major component of the adversary is its scheduling capability. In both randomized and deterministic protocols, the scheduling capabilities of the adversary have a profound effect on system performance. These capabilities may include control over the timing of message delivery and on the rates of processors' internal clocks. The adversary may even vary these rates dynamically, as the protocol runs. Such dynamic control models the case in which a processor is running several protocols concurrently.

As the load on this processor varies, the rate at which it takes steps in any given protocol may change.

The synchronous and the completely asynchronous models capture the two extremes in the degree of control the adversary can possess over the timing of events. In the synchronous case, the adversary has no control over the rates of internal clocks of correct processors. Similarly, the adversary has very limited control over delivery time of messages between correct processors, as all messages reach their destination within a known time limit. However, the adversary may be able to *rush* messages to and from faulty processors. This is discussed further below.

In the completely asynchronous case the adversary has complete control over the order and timing of message arrival and of the internal clocks. There is no upper bound on message delivery time or on the relative rate of internal clocks. Thus there is no way to distinguish a very slow processor from one that has crashed, just as there may be no way to know if an expected message will eventually arrive or if it was never sent. This implies that asynchronous protocols can usually only be message driven, as opposed to clock driven. The only limitation on the adversary is that all messages sent by nonfaulty processors, of which there are at least $n - t$, must eventually be delivered.

The third type of restriction is usually referred to as the *fault model*. Three archetypical models are widely used for processor failures. The first, most benign one, is the *fail-stop*, or *crash* failure model. In this model, processors always send messages according to the prescribed protocol. A processor fails by crashing at some arbitrary time and from then on sends no messages. Note that this model does not guarantee "atomic broadcast" in that some of the messages sent at the time of failure may arrive while some others may not arrive. A more severe type of failure is the *omission fault*. In this model, all processors send messages according to the prescribed protocol at all times, but some messages sent by faulty processors might fail to reach their destinations. This models the case in which a processor has an intermittently faulty transmitter. The strongest failure model is the *Byzantine* failure model, in which faulty processors can completely deviate from the protocol. In particular, they can send wrong and conflicting messages, so as to cause maximum confusion and disarray to others.

All three failure models can be viewed uniformly as cases of processors that cease to follow their prescribed protocols and

instead start following an adversary protocol. The processor failure model then restricts what messages could possibly be sent by faulty processors. The adversary determines which messages are sent by faulty processors subject to these restrictions.

A popular variant of the Byzantine failure model assumes the availability of an unforgeable *authentication mechanism*. A processor receiving a signed message can convince others that the message originated from the stated sender by displaying the signed text. (See Appendix II for more details). We note that in the case of several protocols in the authenticated Byzantine model appearing in the literature the authors simply assume an authentication mechanism and prove the protocol correct only under this assumption. In a model where faulty processors garble relayed messages at random, error correcting codes can be used for authentication.

The fourth aspect of an adversary concerns the resources to which it has access. These can be of two types: computation resources and information resources. The *computationally unbounded* adversary has no limit to its computing power and can even use nonrecursive strategies. A more limited adversary appearing often in the literature is the *computationally bounded* adversary. This adversary has only probabilistic polynomial time to determine its moves, where the polynomial is in terms of n, the size of the system, and the *security parameter* h.

Correct processors are often considered to be limited to probabilistic polynomial time computations, reflecting the popular view that anything outside this class is not "feasibly" computed. Given a similarly limited adversary, cryptographic techniques may be used to obtain more efficient agreement protocols. These techniques employ cryptographic operators, such as public-key encryptions or pseudorandom sequence generators (see Appendix II). They are based on certain complexity theoretic assumptions, such as the assumption that factoring cannot be accomplished in random polynomial time. These assumptions, while not proved (a proof would imply that $P \neq NP$), are widely believed to be true.

Under certain specific assumptions, such as the factoring assumption above, it is possible to design digital signature schemes in which the forging of signed messages is infeasible [GMR84]. Such signature schemes can be used to implement an authentication mechanism in the presence of a computationally bounded adversary. We remark that, to the best of our knowledge, when used in a system with $t \geqslant n/3$, any authentication mechanism that is based on digital

signatures requires that all n public signature keys be part of the program of each processor.

In defining the adversary one must also specify to which parts of the system the adversary has access. This is what is meant by information resources. If *eavesdropping* is allowed, then the adversary can listen in on the conversations between correct processors. If these lines are secure, then we say eavesdropping is not allowed. Sometimes eavesdropping can be handled by using encryption. However, as we shall see in later sections, introducing cryptography is almost never as simple or straightforward as one would like it to be. And of course, cryptography can be used only in the context of a computationally bounded adversary.

If the adversary has access to the internal states (but never the random tapes) of the correct processors, then we say it is *intrusive*. A protocol in which correct processors employ cryptography to hide information from the adversary cannot tolerate an intrusive adversary. This is because an intrusive adversary has access to all the secret keys of the correct processors. A related, more subtle, issue is the amount of information the adversary may gain about the internal state of an initially correct processor at the instant the adversary makes that processor faulty. One possibility is that the adversary has access to the internal states of corrupted processors. (A processor's internal state may include its entire history.) In another variant considered in the literature, the adversary completely controls the external behavior (e.g., messages sent and received) of the corrupted processors, but it has no access to the internal states. These distinctions have implications in a cryptographic setting, where, for example, the decryption key of a newly corrupted processor may yield information about the state of a second uncorrupted processor.

In addition to deciding which processors to *subvert*, the adversary must specify some behavior for the faulty processors. The time at which these decisions are made is characterized by the *adaptiveness* of the adversary. This is the fifth major component of the adversary.

There are several subtle variations in the assumptions regarding the subversion and control of faulty processors as a function of the history of the system. Interestingly, in the context of Byzantine agreement these issues arise only in randomized protocols. If both the adversary and the processors are deterministic, then the adversary can completely determine the outcome of its strategy. By contrast, a key property of randomized protocols is that the adversary has no way of predicting future coin tosses of correct

processors. Thus, even a computationally unbounded adversary cannot be certain of the outcome of its strategy before the processors' random choices are made. It may therefore be advantageous for the adversary to decide its behavior adaptively, as the random bits are used, so in modeling the power of the adversary it is crucial to specify the extent to which the adversary is adaptive.

To describe some of the common adaptability assumptions we consider only the synchronous case. Adversaries analogous to those discussed below are defined for the asynchronous case whenever applicable. In the *static* model, the adversary chooses which processors will be made faulty during the execution of the protocol before processing begins. The *weakly dynamic* adversary selects at the end of round $r-1$ which processors it will subvert in round r. The *strongly dynamic* adversary, sometimes called the *blocker*, can first activate all processors and examine the messages they wish to deliver at around r. Based on this examination, the adversary can allow some of the messages to be delivered intact, while altering others, thereby subverting the senders of the altered messages. It may be the case that there is a small set of processors the adversary could choose to make faulty that will cause the algorithm to run slowly, but that this set is hard to find. Thus a resource bound adversary may be unable to find this set of processors.

In the static and weakly dynamic adversary models it is necessary to specify whether or not the round r messages of those processors subverted by the end of round $r-1$ can depend on the round r messages of correct processors. If the adversary is allowed to first "wait" for all round r messages to which it has access and only then instruct the faulty processors on their own round r messages, then we say that *rushing* is allowed. If the adversary must choose its round r messages at the termination of round $r-1$, then we say that rushing is not allowed. Rushing is a realistic type of fault whenever there is sufficient uncertainty in message transmission time. The length of a communication round must be chosen as large as the maximum possible transmission time between correct processors, but if a message happens to be delivered to a faulty processor in time less than this maximum, the faulty processor has the opportunity to rush.

2.3. Cost Measures

There are three principal cost measures by which the protocols discussed in this survey are usually compared. The first is sometimes

called *time*. In the synchronous system this is exactly the number of rounds of message exchange and so is often referred to as *rounds*. In asynchronous systems time may signify the maximum length of any chain of messages (think of this as the maximum number of times a message could be forwarded plus one for the original transmission). However, there is something akin to a round even in a completely asynchronous system. Consider a set of n processors running a protocol tolerant to t faults, and let p be a correct processor in this set. If p broadcasts the message for its ith step in the protocol and receives step i messages from only $n - t$ processors, then p cannot safely wait for additional step i messages because all t processors from which it has not heard may be faulty. In this case, p must proceed to step $i + 1$ in its protocol, and we say p has completed round i.

The second principal measure by which agreement protocols are compared is the *resiliency*. This is the maximum number t of faulty processors that can be tolerated when the protocol is run with n processors. The resiliency often has the form $t = (n - 1)/c$, where c is a constant.

The third principal measure is the communication complexity of the protocol, measured either as the total number of bits sent in each round, or as the total number of bits sent in the lifetime of the protocol. In general, the literature had addressed only the question of whether the number of bits grows polynomially or exponentially in n and t. We will not give precise bounds on bits, and unless otherwise stated all the protocols considered in this survey require in each round a number of bits that is polynomial in n and t. In the case of the computationally bounded adversary, where a security parameter h is involved, the number of bits is also polynomial in this parameter. Occasionally in an asynchronous protocol a processor will tag each message with its view of the round number. In this case the number of bits grows (at least) logarithmically with the round number. This is not an issue in any of the asynchronous protocols mentioned in this survey, as all of these are expected to run within a number of rounds that is at most $2^{O(n)}$.

Two other cost measures of interest are the amount of local computation, as a function of n, t, and possibly a security parameter h, that must be performed by each processor at each round, and the number of random bits that must be generated by the processors, either per round or total over the expected running time of the protocol. In all the randomized protocols surveyed here, the local

computation at each round is polynomial in n, t, and h. The question of minimizing the expected number of random bits needed in each execution of the protocol has rarely been addressed in the literature (see [CC84]).

3. BYZANTINE AGREEMENT

There is a rich literature on Byzantine agreement, and several variants of the problem have been studied. In this survey we examine randomized solutions to the following version. Each processor begins with a private binary value. During the course of the computation each correct processor must irreversibly *decide* on a value in $\{0, 1\}$, subject only to the requirements that all correct processors decide on the same value and if all correct processors begin with the same value, then that is the value chosen. (A reduction from multi-valued agreement to binary agreement is given in [TC84].)

We also describe another version, called the *single source Byzantine agreement* problem, because some of the protocols discussed in this chapter employ solutions to the single source version as subroutines. In this version of the problem the intent is for all processors in the system to agree on a message sent by the source, which is itself just another processor in the system. The requirements are that if the source is not faulty, then the value agreed on must be the value sent by the source. However, if the source is faulty then any decision value is acceptable, provided that no two correct processors decide differently. The two problems are intimately related. Indeed, all the results mentioned in this chapter apply to both problems.

The remainder of this section is organized as follows. In Section 3.1 we discuss certain lower bounds for deterministic agreement protocols. This allows us to measure the gains achieved by randomization. In Section 3.2 we discuss the principal original randomized protocols. Improvements and extensions to these protocols are noted in Section 3.3. Sections 3.4 and 3.5 briefly mention some related results.

REMARKS.

1. All randomized protocols discussed in this section require a number of communication bits that is polynomial in n at

each round. Thus, unless the execution runs for more than a polynomial number of communication rounds, the total number of bits remains polynomial.

2. The plethora of models considered in the literature makes comparisons between protocols extremely difficult. Even stating the precise adversary or adversaries to which each protocol is tolerant is cumbersome. The interested reader may find some of these details in Appendix I.

3.1. Lower Bounds for Deterministic Protocols

Tight bounds on time and resiliency for deterministic Byzantine agreement protocols are known for all failure models considered in the literature. Pease, Shostak, and Lamport [PSL80] showed that in the presence of a computationally unbounded adversary with Byzantine faults, t-resilient Byzantine agreement is possible if and only if n, the total number of processors, satisfies $n \geqslant 3t + 1$. As we discuss in Section 5, this bound on resiliency is not amenable to randomization. In other words, even using a randomized protocol, if $n \leqslant 3t$ then sometimes either the requirement that correct processors agree on the decision, or the requirement that if all correct processors begin with the same value then this is the value chosen, must be violated. However, if the adversary is restricted so that faulty processors cannot forge messages of correct processors, i.e., if an authentication mechanism is available, then Byzantine agreement can be reached even if $t = n - 1$ [PSL80]. This result applies a fortiori to all weaker fault models, in which the faulty processors never "try" to relay false information.

In the remainder of this section we discuss two lower bounds for deterministic protocols: a $t + 1$ round lower bound for agreement in synchronous systems and an impossibility result for completely asynchronous systems. Not only can randomization help us cope with these limitations, but as we shall see in later sections, often the same techniques apply to both problems.

The first lower bound on the number of rounds needed to reach agreement was obtained for the unauthenticated Byzantine model by Fischer and Lynch in 1982 [FLy82]. They proved a worst case lower bound of $t + 1$ rounds for any agreement protocol tolerant to t faults. This matches the upper bound of the protocol of Shostak, Lamport, and Pease [PSL80].

In 1982 Dolev and Strong, and, independently, DeMillo, Lynch, and Merritt, extended the Fischer and Lynch result to authenticated Byzantine failures [DS82, DLM82]. Soon afterward, the $t + 1$ round lower bound was extended by Hadzilacos to omission faults [H83], and later by Fischer and Lamport to fail-stop faults [FLa82]. In all but [FLy82] the underlying proof is the same. One begins, for the sake of contradiction, with the assumption that there is a protocol that always terminates in $k \leqslant t$ rounds. The general strategy is then to create a chain of k-round executions, each adjacent pair in the chain indistinguishable to some correct processor, such that the initial conditions force a decision of 0 (all start with initial value 0) in the first execution and a decision of 1 (all start with 1) in the last execution. Because each pair of adjacent executions is indistinguishable to some correct processor, some such processor will be forced to decide both 0 and 1 in the same execution, which is not allowed. Later, in discussing lower bounds on randomized protocols, we will be interested in the length of the shortest chain of scenarios for which the proof goes through. Chor, Merritt, and Shmoys [CMS85] claim that for executions of length $k \leqslant t$ a chain of length at most $2(2\lceil n/\lfloor t/k \rfloor \rceil)^k$ suffices.

The second major lower bound for deterministic protocols is due to Fischer, Lynch, and Paterson [FLP83]. In a seminal paper they proved that in a completely asynchronous system no agreement protocol can tolerate even one fail-stop processor failure. Intuitively, the difference between synchronous and asynchronous systems arises because in the synchronous case if a processor does not hear from some other processor in a given round then it knows that processor is faulty, while in the asynchronous case if it hears $n - 1$ processors it has no way of knowing if the remaining processor from which it has not heard is bad or just slow.

Beginning with the assumption that a completely asynchronous 1-resilient agreement protocol exists, they derive a contradiction by arguing along the following lines. They first observe that the hypothetical protocol must have some "bivalent" assignment I of the initial value. This means there exist two executions of the protocol, both beginning with initial assignment I, such that one execution results in a decision of 1, while the other results in a decision of 0. Fisher, Lynch, and Paterson then argue that in any execution beginning in a bivalent state and resulting in a decision there must be some "critical step" taking the system from a bivalent configuration to a configuration from which only one outcome is

possible. Exploiting the asynchrony of the system, they show that the system can "hide" the critical step by temporarily delaying any messages send during that step. Intuitively, we see that if a processor p takes the critical step but the message system delays its messages, then the other processors may be forced to wait. However, they cannot wait safely because for all they know p could have failed before taking the critical step, in which case the other processors would wait forever. By careful formalization of this intuition, Fischer, Lynch, and Paterson show that for the hypothesized 1-resilient asynchronous agreement protocol there is an infinite execution in which no processor fails, every message sent is eventually delivered, and no processor ever enters a deciding state.

3.2. Randomized Agreement—Principal Results

The impossibility of reaching agreement in an asynchronous system motivated many researchers to explore the feasibility of randomized solutions. The first two randomized agreement protocols appeared in 1983, and were developed by Ben-Or and Rabin, respectively. Rabin's protocol requires certain very strong assumptions, but has low expected running time, even for $t = \Theta(n)$. Ben-Or's protocol requires no such assumptions, but has exponential expected running time for $t = \Theta(n)$. In 1985 Bracha modified a synchronous version of Ben-Or's protocol, dramatically improving the expected running time for synchronous Byzantine agreement. Bracha's protocol is nonconstructive in that it requires the assignment of processors to "committees" satisfying certain conditions. While Bracha showed that almost all assignments of processors to committees of the appropriate size satisfy the desired conditions, no explicit construction is known. In this section we discuss these three principal contributions of Rabin, Ben-Or, and Bracha. As mentioned in the Introduction, all these protocols require "coins" visible to certain (possibly large) subsets of the processors.

In both Rabin's and Ben-Or's work the original goal was to achieve agreement in a completely asynchronous system, but the resulting protocols can be run in the synchronous environment, often with high resiliency. In some instances (Rabin's protocol, Ben-Or's protocol with sufficiently small t, Bracha's protocol) the expected running time of the synchronous randomized protocol is considerably less than the lower bound of $t + 1$ rounds for deterministic protocols. In all these protocols, both synchronous and

asynchronous, the two correctness critieria of Byzantine agreement are never violated: no two correct processors ever decide on conflicting values, and if all correct processors begin with the same value then that is the value chosen. In addition, the probability that some processor has not decided by round r approaches 0 as r approaches infinity. In all of these protocols the earliest round at which all correct processors have decided is a random quantity. Moreover, once a correct processor has reached a decision it may be obliged to continue participating in the protocol for some number of rounds. Both the synchronous and asynchronous versions of all protocols discussed have finite expected running time. However, for simplicity, we do not address the issue of termination in this survey. For the same reason we discuss only the synchronous versions of the protocols.

3.2.1. *Rabin's Protocol*

The major contribution of Rabin's work [R83] is to demonstrate the usefulness of a source of randomness visible to all processors. Let us assume the existence of a procedure COIN TOSS implementing such a source, each invocation of which instantly produces a new random bit visible to all correct processors. Using this COIN TOSS procedure, Rabin's protocol achieves Byzantine agreement in constant expected time.

We assume the system is synchronous and take $n \geq 8t + 1$ for our discussion. Rabin's protocol (Figure 1) described below, runs as a possibly infinite number of iterations of a main loop. (The code is easily modified so that processors halt soon after they decide.)

Figure 1. Code for Rabin's protocol; version for $n \geq 8t + 1$.

```
1. procedure RABIN AGREEMENT(vote):
2. low← ⌊n/2⌋ + t + 1
3. high← ⌊n/2⌋ + 2t + 1
4. do forever
5.    broadcast vote
6.    receive votes from all processors
7.    Let majority value be that value occurring most frequently in the votes received
8.    Let tally be the number of occurrences of majority value
9.    coin ← COIN TOSS
10.   if coin=1
11.      then threshold ← low
12.      else threshold ← high
13.   if tally ≥ threshold
14.      then vote ← majority value
15.      else vote ← 0
16.   if tally ≥ n − t then decide majority value
17. od
```

Throughout execution of the protocol each processor favors a particular candidate decision value, called its *vote*. In the first iteration the vote is the processor's private initial value. However, a processor's vote may change as execution progresses.

If in some iteration k of Rabin's protocol two correct processors differ in their choice of *majority value* computed in Step 7, then regardless of which threshold is chosen the *tally* of no processor exceeds the threshold. In this case all correct processors will choose 0 as the new value of *vote* in Step 15. Thus at least $n - t$ correct processors broadcast 0 in Step 5 of iteration $k + 1$, so all correct processors decide 0 in Step 16. We therefore concentrate on the case in which all correct processors compute the same *majority value* in Step 7.

Observe that the two thresholds *low* and *high* differ by at least t. We say the adversary *foils* a threshold if by having the faulty processors send different values to different correct processors it causes the value of *tally* to exceed the threshold in the view of at least one correct processor, while simultaneously causing the value of *tally* to fall short of the same threshold in the view of a different correct processor. Because the two thresholds differ by at least t, the adversary can potentially foil only one of *low* and *high* in each iteration of the loop. Thus in each iteration the comparison chosen by the procedure COIN TOSS is foiled with probability at most 1/2, so the number of iterations to an unfoiled comparison is a geometrically distributed random variable with expected value 2.

Consider an execution of Rabin's protocol in which for some iteration k the threshold chosen in k is not foiled. In Steps 13–15 all correct processors choose the same value v of *vote*, in iteration k. Thus in iteration $k + 1$ every correct processor receives at least $n - t > high > low$ votes for v in Step 6, sets *majority value* $\leftarrow v$ in Step 7, and sees in Step 13 that *tally* exceeds whichever threshold is chosen. Further, if in any iteration a correct processor sees *tally* $\geqslant n - t$, then in the view of all other correct processors *tally* $\geqslant high$, so all correct processors choose the same value of *vote*, independent of the threshold chosen, and whether or not the threshold is foiled. Combining these two observations, we see that if the threshold chosen in iteration k is not foiled then from iteration $k + 1$ on no correct processor changes its vote, and all the votes agree. We also note that regardless of the coin, if the correct processors begin the protocol with the same initial votes, then agreement is reached within a constant number of iterations.

To implement the COIN TOSS procedure Rabin suggests using a trusted dealer to precompute a random sequence of 0s and 1s (coins) and to distribute the result to all participants before execution of the algorithm begins. To achieve the distribution of the coins in such a way as to keep the random sequence unpredictable to the adversary, Rabin uses Shamir's [S79] technique for sharing a secret with threshold t. This is a method for sharing a secret s among n processors in such a way that no t or fewer can reconstruct s, while any set of at least $t + 1$ processors can do so (see Appendix II).

In general, in order to be used in implementing Rabin's algorithm, the COIN TOSS procedure should satisfy two requirements: the coin should be unpredictable (otherwise the adversary has an advantage in choosing which threshold to foil), and all processors should see the same random bit in each iteration (so that they compare their tallies to a common threshold). However, as we now explain, Rabin's result is actually quite robust, and both requirements can be considerably relaxed.

Let $M = \lfloor n/2 \rfloor + t + 1$. Suppose the coin tossing procedure only guaranteed that at leat M correct processors see the same value of the coin. Even under this weaker assumption, if the coin is unpredictable to the adversary, then a slight modification allows agreement to be reached in a constant expected number of rounds. To see this, consider a particular iteration. After executing the coin toss in step 9, each processor broadcasts its value of COIN and takes as the outcome of the coin toss that value received from a majority ($> n/2$) of the participants. If initially M correct processors received the same value from COIN TOSS, then after the additional round all correct processors see the same value, a situation we have already analyzed. Further, if in a given iteration fewer than M correct processors see the same value of the coin, then no damage is done. This is because a processor decides on a value v only if *tally* is so large that in view of all correct processors *tally* > *high*, so *tally* exceeds any threshold chosen, even if different processors choose different thresholds.

These observations lead us to the following concepts, which are treated more rigorously in Section 4.1. Loosely speaking, a *distributed coin flipping* protocol is a procedure by which a set of processors combine their private sources of randomness to obtain a random value visible to all or most of the participants. We say a t-resilient coin flipping protocol achieves a *persuasive coin* if for some $\varepsilon > 0$ in every execution of the protocol the probability that

at least $\lfloor n/2 \rfloor + t + 1$ correct processors see the outcome of the execution as 0 and the probability that at least $\lfloor n/2 \rfloor + t + 1$ correct processors see the outcome as 1 both exceed ε. This ε may be a function of n and t, but it is usually a constant. Notice that in general the sum of the probabilities that M correct processors all see one of the two possible outcomes is less than 1. Thus in any particular execution it may be the case that no set of $\lfloor n/2 \rfloor + t + 1$ correct processors see the same outcome. An execution of a persuasive coin protocol in which at least $\lfloor n/2 \rfloor + t + 1$ correct processors actually do see the same outcome is called a *visible flip*.

Let $\mathscr{C}$ be a t-resilient protocol for achieving a persuasive coin in a system of n processors. Let $T(n, t)$ be the run time and E its expected value of $\mathscr{C}$. [In the protocols discussed in this survey $T(n, t)$ is not random, but in principle it could be.] Let ε be the minimum of the probabilities of obtaining a "visible 1" and a "visible 0" in an execution of $\mathscr{C}$. Then Rabin's result implies that Byzantine agreement can be achieved in expected time $O(E\varepsilon^{-1})$ by using $\mathscr{C}$ as the COIN TOSS procedure. Later we will discuss various protocols for flipping persuasive coins that do not make use of a trusted dealer.

3.2.2. *Ben-Or's Protocol*

As mentioned earlier, Ben-Or designed a randomized agreement algorithm [Be83] around the same time as Rabin obtained his results. The two algorithms are similar in flavor and therefore we will not describe Ben-Or's protocol in detail. The principal difference between the two protocols is in the way in which the COIN TOSS procedure is implemented. Ben-Or was the first to give a truly distributed implementation of a coin flipping procedure.

Ben-Or's implementation of COIN TOSS is very simple. Each processor flips a private coin and uses the outcome as the "result" of the coin flipping procedure. As the coins of correct processors are flipped independently, the number of 0s and the number of 1s flipped by correct processors are usually expected to be roughly equal [within $O(\sqrt{n})$ of each other]. However, occasionally a large majority, $\lfloor n/2 \rfloor + t + 1$, of the correct processors will flip the same value. When this happens the effect is that of a visible flip, regardless of the actions of faulty processors. While the probability of

such an event is exponentially small, $2^{-\Omega(n)}$ for $t = \Omega(n)$, it is still positive. Therefore this COIN TOSS procedure can be used even in the asynchronous case to reach agreement in finite (exponential) expected time, in the presence of a computationally unbounded, strongly dynamic, intrusive, eavesdropping adversary.

When $t = O(\sqrt{n})$ the expected time to achieve a visible flip in Ben-Or's protocol is $O(1)$. This is because the probability that at least $n/2 + c\sqrt{n}$ correct processors will flip the same value is proportional to $1/k_c$, for some constant k_c depending only on c and not on n. This analysis depends on the fact that coins of correct processors are sufficiently unbiased and independent. This condition holds trivially when the n coins are flipped locally, as in Ben-Or's protocol. More generally, it is necessary and sufficient that the bias of each private coin satisfy $|Pr(0) - Pr(1)| = O(1/\sqrt{n})$, where $Pr(x)$ denotes the probability that the private coin takes value x, provided these coins are independent. These conditions play an important role in Bracha's protocol, which we describe next.

3.2.3. *Bracha's Protocol*

In 1985 Bracha [Br85] published a linearly resilient synchronous agreement protocol that required no trusted dealer and whose expected running time dramatically improved on the expected running time of Ben-Or's protocol when run with similar resiliency. (Bracha did not address the problem of asynchronous agreement. Ben-Or [Be85] later adapted Bracha's protocol to run asynchronously, at a certain cost. See Section 3.3.) Specifically, Bracha obtained a method of translating a system of n processors of which nearly one-third are faulty [$t < n/(3 + \varepsilon)$, where $\varepsilon > 0$ is any constant], to a system of m processors, of which up to $\sqrt{m}$ are faulty.

Ben-Or's synchronous protocol can then be simulated on the new system. The translation is accomplished by creating m *compound virtual processors*, or *committees* for brevity, each composed of a set of s actual processors, for suitable choices of m and s. A single actual processor may belong to several committees. The intent is for each committee to simulate a single processor in a system of size m, where the simulated system is running Ben-Or's protocol, although this idea has broad applicability.

Let us say a committee is faulty if one-third or more of its members are faulty. Otherwise it is said to be *good*. At the heart of

Bracha's result is a counting argument that there exists an assignment of n actual processors to $m = O(n^2)$ committees such that for all choices of $t < n/(3 + \varepsilon)$ faulty processors, at most $O(\sqrt{m})$ committees are faulty, and each committee has size $s = O(\log n)$. We will refer to such an assignment as a *Bracha assignment*. Bracha actually showed that almost any assignment of n processors to m committees of size s has the desired property, but at this writing no efficient scheme (i.e., time polynomial in n) is known for generating a Bracha assignment.

Because Ben-Or's protocol has constant expected running time when $t = O(\sqrt{n})$, the expected running time of Bracha's protocol is linear in the time it takes his system to simulate a single round of Ben-Or's protocol. In order to do this, the actual processors in a given committee must be able to agree on the messages received and messages to be sent by the committee. Most importantly, good committees must be able to flip reasonably unbiased coins, global to that committee and independent of the coins of other good committees. (A precise definition of "bias" is given in Section 4.1.) The running time of Bracha's protocol is actually dominated by the cost of flipping a coin in each committee. Using deterministic Byzantine agreement as a subroutine in each committee, this running time is dominated in turn by the maximum number of faulty processors within the good committee. For $t = \Omega(n)$, the number of faulty processors in good committees is $\Theta(s)$, and so the running time is linear in s, the size of the committee.

Recall that for Ben-Or's protocol to run in constant expected time the bias of the private coin of a correct processor should not exceed $O(1/\sqrt{n})$, where n is the number of processors. Similarly, in the simulation of Ben-Or's protocol the bias of a committee's coin should not exceed $O(1/\sqrt{m}) = O(1/n)$ in order for Bracha's protocol to run in expected time $O(s) = O(\log n)$, and the coins of good committees should be mutually independent. In Section 4 we discuss some approaches to obtaining the high quality coins necessary for the simulation.

All protocols for this problem (and hence Bracha's agreement protocol) of which we are aware require either that the adversary cannot eavesdrop or that the adversary is computationally bounded.

We note that it is not possible to apply Bracha's technique recursively to the Byzantine agreement within the various committees in order to speed up the entire procedure to $o(\log n)$ expected time.

This is because all but at most $O(\sqrt{m})$ committees must agree internally on the values of their coins. So even if the expected time to agreement within each individual committee could be reduced to a constant, the expected time for $m - O(\sqrt{m})$ committees to terminate would still be $O(\log m) = O(\log n)$. (The expectation of the maximum of k identically distributed random variables is $\log k$ times the expectation of each variable.) This represents no improvement on the time needed for all committees to reach internal agreement deterministically.

3.3. Randomized Agreement—Improvements and Modifications

In this section we briefly describe various improvements to and modifications of the results of Rabin, Ben-Or, and Bracha. These include improvements in resiliency and/or running time, approaches toward the generation of a Bracha assignment, an asynchronous version of Bracha's protocol, and an agreement protocol that achieves constant expected time after an initial preprocessing stage without assuming a trusted dealer.

We first discuss improvements in resiliency. Bracha's protocol is almost optimally resilient. This is not true of Ben-Or's asynchronous and Rabin's synchronous and asynchronous protocols. However, these bounds were tightened by Bracha [Br84] and by Toueg [T84].

Ben-Or's original asynchronous protocol requires $t < n/5$. The same protocol run synchronously requires only $t < n/3$, the best possible resiliency for unauthenticated Byzantine agreement, with or without randomization [PSL80, KY84]. Intuitively, the difference arises because in the synchronous case if a processor hears from only $n - t$ processors in a given round then it knows that the ones from which it does not hear are faulty, while in the asynchronous case if it hears from $n - t$ it has no way of knowing if the t from which it had not heard are bad or just slow. In 1984 Bracha closed this gap, obtaining an asynchronous protocol for any $t < n/3$ [Br84]. This resiliency is optimal for asynchronous systems, even if authentication is assumed [BT83]. The papers of Bracha and Toueg [BT83, Br84, T84] introduced to the Byzantine literature a very important methodological technique: identification and implementation of appropriate communication primitives. In Bracha's case, the net result of these primitives is to effectively restrict the behavior of malicious processors to that of fail-stop processors (possibly with faulty private coins). Bracha's *broadcast* primitive

guarantees that the same messages are received by all processors, although they may be received at different times. The *validate* primitive allows a processor to verify that a message received via the *broadcast* primitive should indeed have been sent if the broadcaster was following its protocol correctly. A similar broadcast primitive, called *echo-broadcast*, was developed by Bracha and Toueg in 1983 [BT83]. The communication behavior of a faulty processor is limited by the *echo-broadcast* primitive so that contradictory messages are not received by different parties. That is, a message sent by a faulty processor may be accepted by some correct processors and not by others, but it will never be the case that two correct processors accept conflicting messages from the same faulty processor. This primitive was used by Toueg [T84] to improve the resiliency in the model of Rabin (assuming a trusted dealer). See Appendix I for specific bounds. We remark that Rabin's protocol requires that only the trusted dealer be able to unforgeably sign messages, while in Toueg's protocol this ability is required of the processors as well. The approach of [Br84, T84] has yielded other elegant distributed protocols (cf. [ST84, ST85, TPS85]).

We now consider improvements in expected running time in the presence of a computationally unbounded, eavesdropping adversary. Although Ben-Or's protocol tolerates such an adversary, when run in a synchronous environment its exponential expected running time yields no improvement over the running times of various deterministic algorithms. The only protocol of which we are aware for this model was designed by Chor and Coan [CC84]. In contrast to Ben-Or's protocol, the protocol of Chor and Coan cannot tolerate a strongly dynamic adversary, but it can tolerate weakly dynamic behavior. By careful selection of which processors toss coins in each iteration, Chor and Coan were able to obtain expected running time $O(t/\log n)$ while maintaining the optimal maximum resiliency for a computationally unbounded adversary, $t < n/3$. Their approach is to partition the processors into $n/\log n$ disjoint groups of size $g = \log n$. Processing proceeds in a round robin fashion, with epochs belonging to each group in turn. In a given epoch only members of the owning group flip coins. Each processor in the group broadcasts the result of its private coin flip to the entire network. Each processor (including members of the owning group) computes the majority of the values it has received from the members of the owning group and takes that majority value to be the result of the COIN TOSS. Note that if the private

coin flips of the members of the group are fairly evenly split, then the faulty processors can either cause different correct processors to see different outcomes of the COIN TOSS or to bias the outcome in a direction of their choice. However, if fewer than half the processors in a given group are faulty, then there is a reasonably large probability $[\Omega(1/\sqrt{n})]$ that $g/2 + 1$ correct processors will flip the same value. In this case the majority is determined regardless of the actions taken by the faulty processors. With at most t faulty processors there can be no more than $2t/g$ groups for which a majority of the members are faulty. Thus, after at most $2t/g$ epochs an epoch belonging to a "good" group is reached. As the expected number of good groups that have to be used until agreement is reached is rather small $[O(\sqrt{n})]$, the overall expected running time is dominated by the time to reach the good groups, which is $O(t/g) = O(t/\log n)$.

We now discuss three results concerning the generation of Bracha assignments. Bracha's result is considered nonconstructive because it is not known how to explicitly generate an assignment satisfying the desired conditions, except by brute force. It is not even known how to efficiently verify that a given assignment is a Bracha assignment (a co-NP statement). Even though almost all assignments of actual processors to committees of the appropriate size are Bracha assignments, the explicit construction of a correct assignment is an interesting open problem.

Alon has made some progress toward efficient generation of Bracha assignments [A85], though with larger committee size. In his construction m, the number of committees is $O(n)$ and each committee has size $s = \Theta(\sqrt{n})$. Specifically, Alon's construction is based on the projective plane and yields systems of size $n = p^2 + p + 1$, where p is any prime. Every actual processor corresponds to a point in the plane. Every committee corresponds to a line, and thus is composed of $s = p + 1 = \Theta(\sqrt{n})$ actual processors. Alon shows this yields a Bracha assignment in which, if at most $n/(3 + \varepsilon)$ of the actual processors are faulty, then at most $O(\sqrt{m})$ committees are faulty. When Bracha's protocol is run on Alon's construction with appropriately chosen coin subroutine, the expected running time is $\Theta(\sqrt{n})$, because the committee size s is $\Theta(\sqrt{n})$.

To our knowledge no further progress has been made on explicit constructions of Bracha assignments. However, a method for distributively generating an assignment that with high probability is a Bracha assignment was obtained by Feldman and Micali [FM85],

eliminating the need for the Bracha assignment to be distributed prior to the protocol execution. This protocol uses a deterministic agreement algorithm as a subroutine, and therefore requires $\Omega(t)$ preprocessing time.

Each processor secretly generates a string of the appropriate length to describe a Bracha assignment, runs single-source Byzantine agreement on an encryption of the string, and then reveals the string. The bitwise exclusive-or of the revealed individual strings (some processors may refuse to reveal) is used as the Bracha assignment. One initially confusing point about the Feldman and Micali work is that on the one hand the processors must agree on a random assignment to committees by choosing a random string of sufficient length, while on the other hand agreeing on even a single random bit is difficult. This apparent conflict is resolved by observing that, although the faulty processors have a lot of influence on the chosen string, so it is not "truly random," their influence rarely suffices to force the outcome to be a bad assignment of processors to committees. In fact, the faulty processors can cause to be chosen any one of the 2^t strings that are the bitwise exclusive-or of all $n - t$ good processors' strings with some subset of the remaining t strings. However, the bad strings are so few that the probability that any of these 2^t strings are actually bad is still small. For the probabilities to work out, it is necessary that correct processors pick truly random and mutually independent strings, while the rest are chosen arbitrarily, but are independent of the strings of correct processors. Some additional mechanism is required to enforce this independence.

The last extension related to Bracha assignments is a generalization of his counting argument. In 1986 Dwork, Shmoys, and Stockmeyer [DSS86] obtained a distributed coin flipping protocol whose expected time to a visible flip is $O(1)$ provided the number of faults does not exceed $O(n/\log n)$ (see Section 4). When used as the COIN TOSS procedure in either of Rabin's or Ben-Or's protocols this yields an agreement protocol with the stated resiliency whose expected running time is $O(1)$. Generalizing Bracha's counting argument, Dwork, Shmoys, and Stockmeyer show the existence of an assignment of n processors to $m = O(n \log n)$ committees of size $O(\log \log n)$ so that for any choice of fewer than $n/(4 + \varepsilon)$ faulty processors at most $O(m/\log m)$ committees are bad. Simulation of the constant time [DSS86] agreement protocol on the resulting

system yields a linearly resilient protocol whose expected time to agreement is $O(\log \log n)$.

Bracha's result has also been extended to the asynchronous environment. Paying slightly in both resiliency and number of rounds of message exchange, Ben-Or obtained an asynchronous adaptation of Bracha's protocol, requiring $t < n/7$ and running in expected time $O[\log n(\log \log n)^c]$, where c is a constant independent of n and t [Be85]. The protocol was presented in the model without eavesdropping. Ben-Or's adaptation is based on two observations:

1. There exists an exponential time asynchronous Byzantine agreement protocol requiring $n > 3t$ in which each processor executes the COIN TOSS procedure simply by flipping its private coin (cf. [Br84] discussed above).
2. For $n > 4t$, a modification of a distributed coin flipping protocol due to Yao [Y84] gives a t-resilient protocol for a committee to flip a coin visible to all members of the committee, using as a subroutine any (exponential time or faster) protocol for Byzantine agreement among n processors. (See Section 4 for the details of Yao's protocol.)

Recall that Bracha proved the existence of an assignment of n processors to $m = O(n^2)$ committees of size $s = O(\log n)$ so that for all choices of fewer than $n/(3 + \varepsilon)$ faults all but $O(\sqrt{m}) = O(n)$ committees are "good." That is, the resulting number of faults in the simulated system of size m is $O(\sqrt{m})$. Therefore, running the asynchronous version of Ben-Or's protocol [Be83] in the new system yields agreement in expected time $O[T(s)]$, where $T(s)$ is the time needed for all but $O(\sqrt{m})$ committees to simulate message transmission and receipt, and to flip a relatively unbiased coin visible to all members of the committee. By (1) and (2), each individual committee can perform these tasks in expected time $O(2^s) = 2^{O(\log n)} = n^{O(1)}$. Moreover, the expected time for all $m = O(n^2)$ committees to perform these tasks is $O[\log (n^2)n^{O(1)}]$, which is also $n^{O(1)}$. Thus this simple modification of Bracha's protocol yields asynchronous Byzantine agreement in polynomial (in n) expected time and message complexities.

To further improve the expected running time from $n^{O(1)}$ to $O[\log n(\log \log n)^c]$, Ben-Or employs a bootstrapping procedure that is based on the observation that Bracha's result can be applied recursively inside the committees, at a multiplicative cost of

$O[(\log \log n)^c]$. One interesting feature of Ben-Or's protocol is that the processors prepare any polynomial (in n) number of coins in a preprocessing stage, which can then be used to run additional agreements at an expected cost of $O(\log n)$ time per agreement. This feature yields a substantial saving in the amortized cost of internal agreement among lower level committees, which in turn speeds up the expected time for agreement at the top level of the recursion.

This idea of preparing many coins at once to be released later as needed was also used by Feldman and Micali [FM85]. They obtained an elegant method to generate "many coins at the cost of one," based on cryptographically strong pseudorandom number generators (see Appendix II). Some details of this method are discussed in the next section. After a preprocessing stage whose cost is dominated by the cost of establishing pseudorandom number generators inside every good committee, any (polynomial) number of random coins are available. These coins may then be released when desired. In particular, they can be released one per simulated iteration of Ben-Or's protocol. Thus, once preprocessing is completed, the Feldman and Micali protocol can achieve any (polynomial) number of agreements in constant expected time per agreement.

3.4. Weaker Failure Models

Our discussion until this point has assumed Byzantine faults. Chor, Merritt, and Shmoys explored several models in which the message system is the only adversary [CMS85]. Thus, in their models all processors follow the protocol correctly but the adversary can selectively suppress messages. Note that any protocol designed for such a model will a fortiori operate in the model with crash failures, since the adversary can simulate a crash failure by suppressing all messages sent by a given processor. Chor, Merritt, and Shmoys present a protocol resilient to an adversary that chooses whether or not to suppress or "block" a message according to the message contents; hence the term "blocker" to describe the strongly adaptive adversary.

Chor, Merritt, and Shmoys' very natural approach is to have each processor privately flip two coins. The first, called its *bit*, is just an unbiased coin. The second coin is biased so that the outcome is 1 with probability $1/n$ (where value 1 means "I volunteer to be a temporary leader"). Note that when all n processors flip their biased

coins the expected number of 1s (volunteers) is exactly 1. Furthermore, the probability that exactly one processor will volunteer is $\geqslant 1/e$. Once each processor has flipped its two coins it broadcasts the resulting pair of values. In the best case (which occurs with probability greater than $1/2e$) this procedure elects a nonfaulty unique leader. In this case the outcome of the procedure is the bit of the leader. Clearly, this procedure yields a persuasive coin with $O(1)$ expected time to obtain a visible flip. Chor, Merritt, and Shmoys handle the case in which the adversary can block (choose which messages to suppress according to the contents) by encrypting the messages and proving rigorously that unless the adversary can decrypt, the problem is reduced to the weakly adaptive case. A careful analysis shows that the same approach works also in the asynchronous case for $t = \theta(n)$.

3.5. Probabilistic Adversaries

We briefly mention some results in which the adversary is assumed to exhibit certain probabilistic behavior. This probabilistic behavior defines a probability space in which the expected cost of both deterministic and randomized protocols can be analyzed. The motivation for these models is that the worst case assumptions are frequently too pessimistic, and phenomena like failures or delays are often randomly distributed.

The first to examine probabilistic adversaries was Reischuk [Re85]. He considered the model of a uniform static adversary, where all subsets of t processors are equally likely to fail. The adversary is computationally unbounded, and failed processors can exhibit malicious behavior, as is usually the case for Byzantine faults. Reischuk obtained a deterministic Byzantine agreement protocol that terminates in a constant expected number of rounds. The randomized protocol of [CC84] achieves similar bounds in this model.

A different type of probabilistic adversary was studied by Bracha and Toueg [BT83], who developed a protocol essentially identical to Ben-Or's [Be83] but without randomization. Their probability space models an asynchronous environment in which message delays are independent identically distributed random variables. Under the assumption that there exists an ε such that for any two processors p and q, and any round s, p receives a message from q (among the first $n - t$ messages) in round s with probability at least

ε, they prove that the probability of nonconvergence (no decision) after k rounds approaches 0 as k approaches infinity. They obtain protocols with $t < n/2$ and $t < n/3$ for the fail-stop and Byzantine cases, respectively, and prove the resiliency is optimal in both cases.

4. DISTRIBUTED COIN FLIPPING

In Section 3 it was shown that distributed coin flipping may be used in two ways to expedite Byzantine agreement: either directly, in implementing the protocol of Rabin, or indirectly, inside the committees of Bracha's protocol. The two types of coin flipping protocols have very different requirements. For example, as discussed in Section 3.2, the coin in Rabin's protocol need only be persuasive, and it may have a high bias [$O(1)$ expected time is achieved if the bias is constant, independent of n]. By contrast, the committee coins in Bracha's protocol must have small bias [$O(1/\sqrt{m})$]. As another example, in order to keep the overall computation polynomial in the number of processors, the COIN TOSS procedure employed when implementing Rabin's protocol should require only polynomial computation and communication in each round. However, because the committees in Bracha's protocol are small [size $O(\log n)$], the work performed in each committee may be exponential in the size of the committee.

These two types of coin flipping also have different termination requirements. For the persuasive coin it suffices that in every execution there is a nonzero probability that a substantial majority of the processors see the same outcome. In the committees of Bracha's protocol all processors must see the same outcome with certainty. For this reason, we call the second type of coin a *global coin*.

In this section we describe solutions to the persuasive and global coin flipping problems. We consider only Byzantine failures (the only example known to us of a persuasive coin for more benign fault models has already been discussed in Section 3.4). Section 4.1 contains definitions and other preliminaries. Some protocols for persuasive coins are discussed in Section 4.2. A protocol for global coin in the presence of a computationally unbounded adversary that cannot eavesdrop is discussed in Section 4.3. Some of the subtleties arising in cryptographic solutions to the distributed coin flipping problem are discussed in Section 4.4. Global coins in the presence of a computationally bounded eavesdropping adversary are discussed in Section 4.5. Section 4.6 deals with an

"approximate" global coin in which the required communication grows as the inverse of the bias, and a polynomially small bias can be achieved at a "reasonable" cost. In Section 4.7 we briefly mention some results on achieving independence, and application to the problem of flipping a global coin.

4.1. Preliminaries

We consider, as usual, systems of n processors, of which at most t are faulty. Loosely speaking, a *distributed coin flipping protocol* is a procedure for combining the individual random sources of the processors in the system to obtain a source of randomness visible to a "large" number of the participants. We assume that each processor has a special *result* register that is initially empty. During an execution of a distributed coin flipping protocol the processor may irreversibly write a value in the result register, indicating the processor's views of the outcome of the distributed coin flip. If on termination of the execution the value v is written in the result register of some processor p, then we say that p *sees* v as the outcome of the flip. We now specify more precisely the two types of coin flipping protocols used in the agreement protocols described in Section 3.

Let M (for "large majority") satisfy $M = \lfloor n/2 \rfloor + t + 1$. A distributed coin flipping protocol yields a *persuasive coin* in an environment (set of adversary retrictions) $\mathscr{R}$ if there is an $\varepsilon > 0$ such that for every execution of the protocol

$$\inf_{\mathscr{A}} \Pr_{\mathscr{A}}(\text{at least } M \text{ correct processors see } 0) \geqslant \varepsilon$$

and

$$\inf_{\mathscr{A}} \Pr_{\mathscr{A}}(\text{at least } M \text{ correct processors see } 1) \geqslant \varepsilon,$$

where $\mathscr{A}$ varies over all adversary strategies satisfying restrictions $\mathscr{R}$, and $\Pr_{\mathscr{A}}(E)$ is the probability that event E will hold when the given protocol is executed with $\mathscr{A}$. The probability space is that of all internal flips of all processors and of the adversary.

A distributed coin flipping protocol yields a *global coin* in environment $\mathscr{R}$ if in every execution in which the adversary satisfies restrictions $\mathscr{R}$ all correct processors see the same outcome (hence

"global"), and there is an $\varepsilon > 0$ such that

$$\inf_{\mathscr{A}} \Pr_{\mathscr{A}}(\text{outcome} = 0) \geq \varepsilon$$

and

$$\inf_{\mathscr{A}} \Pr_{\mathscr{A}}(\text{outcome} = 1) \geq \varepsilon,$$

where the range of $\mathscr{A}$ and probability space are as above. When the coin flipping protocol is run by a subset of the processors in the system, and each processor in the subset sees the same outcome, we say that the coin is *global to the subset*.

Each execution of a global coin protocol yields a well-defined outcome, while an execution of a persuasive coin need not. We define the *bias* of the coin when run with adversary strategy $\mathscr{A}$ to be

$$\max\{|\tfrac{1}{2} - \Pr_{\mathscr{A}}(\text{outcome} = 0)|, |\tfrac{1}{2} - \Pr_{\mathscr{A}}(\text{outcome} = 1)|\},$$

and the *bias* of the coin as the supremum of this quantity over all adversary strategies $\mathscr{A}$.

We note that when cryptographic protocols are used in a global coin flipping protocol, it is unreasonable to expect the resulting coin to be totally unbiased and unpredictable. This is because even though the adversary is limited to time polynomial in the security parameter, it may be very lucky in guessing the secret keys of correct processors. Thus, the best one can hope for is that the adversary can control or predict the outcome of the coin with probability that is subpolynomial in the security parameter.

The protocols for achieving a global coin described in this section use deterministic Byzantine agreement as a subroutine in order to ensure that all correct processors agree on the outcome. As discussed in Section 3.1, this approach therefore requires $t + 1$ rounds of communication. Broder and Dolev [BD84] showed that in the worst case one could do no better. That is, they proved that any t-resilient protocol to achieve a global coin requires at least $t + 1$ rounds of communication in the worst case. Their technique is essentially the same as that of Dolev and Strong [DS82], discussed in Section 3.1. This lower bound may at first glance be interpreted as implying that a protocol for a distributed coin cannot be used to expedite

Byzantine agreement. However, the lower bound does not apply to persuasive coins, and indeed these can be achieved much more quickly.

In fact, the global coin protocol may be used inside a committee in a Bracha construction. Because the committees are small [$O(\log n)$ in [Br85], and $O(\log \log n)$ in [DSS86]] the time needed to obtain coins global to the committees is also small. Only one additional round of communication is needed to combine the coins global to the committees to obtain a persuasive coin for the entire system.

A final important point is the *independence* of coins global to different subsets. This notion is crucial, for example, in implementing Bracha's protocol, although it was generally overlooked in many of the original papers. To see this, consider two good committees in the network. Suppose the two committees run instances of a global coin flipping protocol (global to each committee respectively) concurrently. Denote the two outcomes by C_1 and C_2. Let us assume that the global coin flipping protocol produces perfect unbiased coins. Even so, it may be the case that C_1 is always equal to $1 - C_2$, without contradicting the fact that each coin is perfectly unbiased. One solution to this problem is to serialize: first obtain C_1 and then obtain C_2. However, serializing is a costly solution in terms of time and would render the protocol of Bracha very slow, as there are $O(n^2)$ committees. What is required is a mechanism for ensuring that concurrent executions of coin flips, global to the various committees, are mutually independent, or at least "almost mutually independent." We do not attempt to present a formal definition of this last notion, but will examine it when discussing the various global coin flipping protocols.

4.2. Persuasive Coins

Ben-Or's Byzantine agreement protocol discussed in Section 3.2 obtains a persuasive coin by having all the correct processors flip their private coins independently. As mentioned earlier, the expected time to achieve a visible flip with constant bias ($< \frac{1}{2}$) in this protocol is $O(1)$ only if $t = O(\sqrt{n})$.

In this section we discuss a protocol of Dwork, Shmoys, and Stockmeyer [DSS86] for achieving a constant bias persuasive coin in $O(1)$ expected time provided $t \leqslant cn/\log n$ for a particular constant c. Applying this technique to Rabin's or Ben-Or's protocol

yields an $O(n/\log n)$-resilient agreement algorithm with constant expected running time. Like Ben-Or's protocol, the protocol of Dwork, Shmoys, and Stockmeyer can tolerate a linear number of faults at the cost of exponential time. However, applying a variation of Bracha's construction to their agreement protocol yields a linearly resilient protocol that requires only $O(\log \log n)$ expected time.

[DSS86] actually contains three protocols for achieving a persuasive coin, a basic coin protocol and two strengthened versions, designed to handle different types of adversaries (see Appendix I). We discuss only the basic protocol.

The basic coin protocol is designed to tolerate the *lock-step adversary with erasing*. In this model the adversary is divided into three parts. At each round r of computation the first part produces a set F_r of processors to be made faulty in round r. The second part produces the (potentially malicious) messages to be sent by processors in F_r. The first and second parts are independent of the round r messages of the correct processors (hence, "lock-step"). The third part, called the *eraser*, is then given the round r messages of the correct processors, and it produces a subset of F_r's messages to be erased. Without the eraser, this model would be the weakly dynamic model without rushing. The eraser is introduced because proving that the basic persuasive coin alogirithm is correct with the eraser facilitates correctness proofs for the two stronger adversaries.

ALGORITHM DSS BASIC PERSUASIVE COIN
Algorithm for processor p_i:

1. Randomly choose an integer value v_i uniformly between 1 and n^4 and choose a random bit b_i.
2. Broadcast (v_i, b_i) to all processors.
3. Compute the outcome of the coin:
   - 3.1. Let V be the multiset of values received from all processors (including p_i). If any value occurs more than once in V, then remove all occurrences of that value from V.
   - 3.2. Let W be the values remaining in V, let $w_1, \ldots, w_m$ be the members of W sorted in increasing order, and let $w_{m+1} = w_1 + n^4$.

3.3. Find the j with $1 \leqslant j \leqslant m$ such that $w_{j+1} - w_j$ is maximized; if there are several such j's then pick the smallest. We refer to this (w_j, w_{j+1}) as the *maximum gap* of W with left endpoint w_j and size $w_{j+1} - w_j$.

3.4. Let p_k be the processor that sent value w_j (p_k is uniquely defined because duplicate values were removed from V). We say that p_k is the *leader* chosen by p_i. The value of p_i's coin is the bit b_k sent by p_k to p_i.

The basic persuasive coin protocol works by selecting a "leader" among the n processors and taking as the value of the coin the bit chosen by the leader. Because the bit is chosen and broadcast by the leader before the processor is actually made leader, the adversary can corrupt the leader's bit only by arranging for a faulty processor to become leader.

Consider a single execution of the protocol. This consists of one round of communication. Let F be the set of processors faulty in this round, and let G (for Good) be the complement of F. Let the random variable CORRECT GAP be defined as the maximum gap between the values v_i of processors $p_i \in G$. (For more details, see steps 3.2 and 3.3 of the basic persuasive coin protocol. CORRECT GAP is computed from the values of processors in G exactly as the *maximum gap* is computed from the values W). The *position* of CORRECT GAP is its left endpoint. Note that if the adversary lands no value in the correct gap then all correct processors choose the same leader in Step 3.4, and this leader is a correct processor.

Let us say the position of CORRECT GAP is *bad* for a correct processor p if p receives a value from a faulty processor that lies between the endpoints of CORRECT GAP. Otherwise the position is *good* for p. Once again, let $M = \lfloor n/2 \rfloor + t + 1$. One can show that for some poisitive constant α

$$\inf_{\mathscr{A}} \Pr_{\mathscr{A}}(\text{position of CORRECT GAP is good for at least } M \text{ correct processors}) \geqslant \alpha,$$

where $\mathscr{A}$ varies over all lock-step with erasing adversary strategies in which at most t processors fail, and the probability space is that of all internal flips of all processors. Note that because the variable CORRECT GAP is defined by the values of correct processors (those in G), erasing the value of a faulty processor does not affect the value of this variable. The protocol therefore handles erasing.

REMARK. As mentioned in Section 3.2, the constant expected time $O(n/\log n)$-resilient agreement algorithm of Dwork, Shmoys, and Stockmeyer, can be transformed into a linearly resilient agreement algorithm requiring $O(\log \log n)$ expected time by a variation of Bracha's construction ([Br85]). However, the simulation of the lower resiliency algorithm requires that the global coins generated by the bad committees be independent of the global coins generated by the good committees. This is not required in Bracha's original algorithm. That is because in his case the lower resiliency system was running Ben-Or's algorithm ([Be83]), in which the private coins of the faulty processors are irrelevant, and therefore need not be independent of the private coin flips of correct processors.

4.3. Unbiased Global Coins with No Eavesdropping

The first to study the global coin flipping problem was Yao [Y84]. Yao's solution assumes a model in which the adversary cannot eavesdrop, but is otherwise very powerful: computationally unbounded, capable of rushing, and strongly dynamic. It requires $t < n/4$, but can be improved to $t < n/3$ with authentication. The protocol was designed for a synchronous system (see [Be85] for an asynchronous version). Yao's solution begins by partitioning the n processors into all $\binom{n}{3t+1}$ subsets of $3t + 1$ processors. Because more than two-thirds of the processors in each subset are correct, Byzantine agreement can be run deterministically within each subset without authentication. Note that at least one subset consists solely of correct processors, regardless of when processors are subverted. Such a subset will be called a *pure* subset.

First, every subset agrees on a binary value as follows. Each of its members individually flips a coin (a prrocessor flips a separate, independent coin for each different subset of which it is an element). Single source Byzantine agreement is then run inside the subset of each of these values. (See the introduction to Section 3 for definition of the single source version of Byzantine agreement.) Thus, as each subset contains $3t + 1$ processors, exactly $3t + 1$ separate executions of single source Byzantine agreement are run concurrently with each subset. Each processor in the subset computes the majority value among the agreed on values and takes that majority value to be the subset coin. Because the properties of single source Byzantine agreement guarantee that all correct processors within the subset

see the same set of $3t + 1$ outcomes, all correct processors in the subset agree on the subset coin.

The point in the protocol at which the $\binom{n}{3t+1}$ subset coins are determined is called the *turning point*. After the turning point each processor in the subset broadcasts the value of the subset coin (this is done for each subset of which the processor is a member). Consider a particular subset S and a processor $p \notin S$. Note that because at least $2t + 1$ of the $3t + 1$ members of S are correct, the value received by p from a majority of the members of S will be the actual value agreed on by the members of S. Finally, the global coin is the mod 2 sum of all $\binom{n}{3t+1}$ subset coins.

Yao's protocol has the property that once the subset's coins have been determined there is nothing the adversary can do to change any of them or to prevent them from being revealed to the other subsets. Thus, at the turning point the final outcome is determined and irrevocable. However, because the pure subset contains no faulty processors, and because by assumption the adversary cannot eavesdrop on the conversations between the members of the pure subset, the coin of the pure subset is completely unpredictable to the adversary at the turning point.

To see that Yao's protocol yields an absolutely unbiased coin, one first argues that the coin of the pure subset is uniformly distributed (this is easy), and that it is independent of all other subset coins. The independence of the pure subset's coin follows from the fact that eavesdropping is not possible, and thus the actions of the adversary before the turning point must be carried out with no information about the actions of the processors in the pure subset. Because the mod 2 sum of $m \geqslant 2$ numbers, at least one of which is uniformly distributed and independent of the others, is itself uniformly distributed, absence of bias in the global coin is achieved.

A similar argument can be made to show that when many instances of Yao's coin flipping protocol are concurrently executed, the resulting coins are mutually independent. Thus, when used as the coin flipping subroutine for the committees in Bracha's scheme, the global coins of good committees are mutually independent. Coins of bad committees are not required to be independent of coins of good committees. Thus Yao's global coin procedure satisfies the required conditions for Bracha's protocol, in the model in which the adversary cannot eavesdrop.

We remark that if $t < \sqrt{n}/3$, then Yao's solution is readily implementable with resources that are only polynomial in n. Simply

partition the n processors into $\sqrt{n}$ disjoint subsets of size $\sqrt{n}$ each. The small number of faulty processors guarantees the existence of at least one pure subset, and the rest of the analysis is identical. For $n > (4 + \varepsilon)t$ (for any constant $\varepsilon > 0$), Yao also obtained a non-constructive global coin flipping protocol in which the message complexity is only polynomial in n. It seems that Yao employs a counting argument similar to the one used by Bracha, but we were not able to obtain further details of this version.

The pattern of coin generation where there is a turning point, just before which the coin is completely determined but its value is unpredictable to all processors, and after which the value is revealed to all, regardless of the actions of the faulty processors, appears in other coin flipping schemes.

4.4. Subtleties in Cryptographic Solutions to the Global Coin Flipping Problem

In recent years much progress has been made in the area of the cryptographic protocol design. Nevertheless, the design of efficient and provably correct cryptographic protocols is still largely an art, rather than a routine task. This is certainly the case in the tricky Byzantine environment. Often, unspecified assumptions about the behavior of faulty processors, implicitly used during protocol design, are later found to be violated, leading to the "breaking" of the protocol. However, if all assumptions and details of the model are explicitly specified, it is possible to rigorously prove the correctness of cryptographic protocols. Such proof should be in the form of a reduction. Specifically, one shows that, under the assumption that the adversary can break the protocol, the adversary can be used in a simulation of the protocol to efficiently solve the underlying intractable problem. In this section, we demonstrate some common traps in naive employment of cryptographic techniques by means of a simple-minded example particularly relevant to coin flipping protocols.

Suppose two participants a and b wish to commit themselves to secret binary values, and later, after the secret values are establised, to reveal these values. Their "global coin" will be the mod 2 sum of their individual values. One method that comes to mind is that a and b will first publish their own public encryption keys E_a and E_b, then publish $E_a(v_a)$ and $E_b(v_b)$, thus establishing the secrets, and lastly announce v_a and v_b by publishing the corresponding

decryption keys D_a and D_b. The idea behind such a protocol is that, with a good encryption scheme, each private value is established by, but unpredictable from, its encryption, and thus v_a and v_b should be independent. One obvious problem with the scheme is that a faulty processor may refuse to decrypt. Suppose a decrypts first. If b does not like the value of the final outcome, and therefore refuses to decrypt, then b's action biases the coin. Even if there is some mechanism that forces b to decrypt, problems remain.

Suppose, for example, that rushing is possible, and b simply *replays*, or *mimics* a's actions: First, b publishes the same encryption key as a does, then it publishes the same encrypted value, and lastly it "decrypts" the same way. The protocol is not violated, but v_a and v_b, being identical, are in no way independent (their sum mod 2 is always 0). In more complicated situations, one cannot rule out the possibility that a faulty processor, after seeing the encryption keys $E_1, \ldots, E_{n-t}$ of correct processors, will come up with an encryption key E_f with the property that for any values $E_1(v_1), \ldots, E_{n-t}(v_{n-t})$, the faulty processor can find $E_f(v_f)$ so that v_f satisfies some desired relation with the values $v_1, \ldots, v_{n-t}$ (be equal to their exclusive-or, for example).

In the above examples messages were encrypted with the public keys of the senders. However, difficulties are also encountered if one encrypts messages with the *receiver's* public key. To see this, consider a system of n processors. Each processor chooses a bit, and breaks this bit into shares using Shamir's secret sharing technique (see Appendix II). The shares are then encrypted in the receivers' public keys and broadcast throughout the network. In the next round all correct processors release their decryption keys, and the shares are decrypted. Any processor found to have created its shares incorrectly is disqualified. The intent here is that any set of $t + 1$ processors can reconstruct the bit of any processor, so a faulty processor cannot prevent its bit from becoming known (in the language of the simple minded example, the faulty processors are forced to decrypt). Once again, the value of the coin will be the mod 2 sum of all the constructed bits. What goes wrong here is that a faulty processor can, for example, send good shares of a secret to all but one process, a collaborator. Once the secrets of the correct processors are known the collaborator can choose whether or not to reveal the bad share.

The underlying problem in the above examples is one of forcing the adversary to choose and reveal its values independently of the

values chosen by the correct processors. Some attempts to deal with this problem in full generality are described in Section 4.7.

4.5. Subpolynomially Biased Global Coins

In this section we discuss cryptographic protocols for achieving a global coin whose bias is subpolynomial in the security parameter of the system. All these protocols assume a computationally bounded, eavesdropping adversary, although complete specification of the adversary was not always given. We also discuss the applicability of these protocols when used to obtain coins global to the committees of the Bracha protocol.

The first cryptographic solution to the coin flipping protocol was proposed by Broder and Dolev [BD84]. Their solution is based on a beautiful idea of combining secret sharing with deterministic encryption schemes (specifically, they suggested high exponent RSA encryption [RSA78], defined in Appendix II). The protocol requires $t < n/3$ in the absence of an authentication mechanism, $t < n/2$ otherwise. The basic idea is that each processor p of a special set of $t + 1$ processors independently picks a binary value c_p, and breaks it into n pieces $c_{p,1}, c_{p,2}, \ldots, c_{p,n}$ using Shamir's scheme with threshold t. The ith piece is sent to the ith processor, encrypted under its public key E_i [that is, p sends $E_i(c_{p,i})$ to i]. Single source Byzantine agreement is run on each of the E_i and the encrypted pieces. This is the turning point. At this point faulty processors know nothing about the value of any correct processor, having access only to t pieces of each value. After the turning point, all processors reveal their decryption keys. As at least $t + 1$ correct processors will reveal their keys, the secret c_p and the rest of the pieces can be reconstructed. (It follows from Shamir's shared secret construction that *any* collection of $t + 1$ pieces not only allows the reconstruction of the secret, but also the reconstruction of the remaining $n - t - 1$ pieces.) Finally, because each encryption function E is deterministic, one can check, given x and $E(y)$, whether $x = y$. Thus, given $t + 1$ pieces, it is possible to verify that the rest of the pieces were correctly encrypted.

The coins of processors found to deviate from the scheme are ignored. The global coin is the mod 2 sum of all the remaining c_p. The following intuitive argument suggests that the resulting global coin is at most subpolynomially biased: as the actions of faulty

processors before the turning point cannot depend on the bit of the correct processor, ignoring the values of processors known to be bad does not bias the coin. With c_p picked independently of the rest, the mod 2 sum ought to be unbiased.

To prove the correctness of their protocol, Broder and Dolev make strong assumptions about the existence of certain families of deterministic public key cryptosystems. They do not prove that RSA with high exponent satisfies these assumptions. In fact, Hastad [Ha85] demonstrated that if the protocol of [BD84] is run with sufficiently low exponent RSA, then the adversary can determine the values of correct processors before the turning point, and thus can predict the final outcome of the coin. While Hastad's method is not sufficient for high exponent RSA, and thus does not directly imply that the scheme is insecure for high exponent RSA, we do not believe that high exponent RSA has the desired properties.

In addition, Broder and Dolev were not able to give a correctness proof for their protocol in the form of a reduction as discussed in the previous section. Indeed, Chor [C85] later observed that if the adversary can rush, then by replaying, it is possible to completely bias the coin (the flavor of this attack is similar to those discussed in the previous section).

The next development in the field was a cryptographic coin flipping protocol of Awerbuch, Blum, Chor, Goldwasser, and Micali [ABCGM85], which requires $t < n/3$ in the absence of authentication, $t < n/2$ otherwise. (As we will see in Section 6, if $t \geqslant n/2$ then no coin flipping protocol with a bias subpolynomial in the security parameter can be achieved.) The unauthenticated version of the protocol requires only the assumption that *some* trapdoor function exist. Both versions are similar to the coin flipping protocol of Yao, described above. The intent was to use cryptography to handle eavesdropping.

The authenticated version of the Awerbuch, Blum, Chor, Goldwasser, and Micali protocol starts by partitioning the n processors into all $\binom{n}{t+1}$ subsets of $t + 1$ processors. In each such subset there is at least one correct processor; moreover, at least one subset is pure, in that it contains no faulty processor. In every subset, the lexicographically first processor is called the sender, and the other t processors are called inspectors. A correct processor plays these roles separately and independently for every distinct subset of which it is a member. Again, the general scheme is to have

every subset flip its own coin, and to then take the mod 2 sum of all $\binom{n}{t+1}$ coins as the global coin.

The protocol for every subset consists of a preprocessing stage and a coin-flipping stage. In the preprocessing stage, the sender and every inspector generate (randomly and independently for every subset) an instance (encryption–decryption pair) of a probabilistic encryption scheme. Single source Byzantine agreement is then run among all n processors on each of the encryption keys E_i, where $1 \leqslant i \leqslant n$. With the sender as the source, single source Byzantine agreement is run among all processors on the t-encrypted messages $E_2(D_1), \ldots, E_{t+1}(D_1)$. Note that $E_j(D_1)$ is the encryption of D_1, the decryption key for the sender, in the encryption key of the inspector j. Thus inspector j can actually check that E_1/D_1 is a valid encryption/decryption pair. At the third and last step of preprocessing, each inspector performs this check. If the pair is not valid the inspector *protests*. By running multiple copies of single source Byzantine agreement concurrently, the whole network gets the same view of the protests. In case the last agreement indicates a protest (whether justified or not), the subset is cancelled. Thus a faulty inspector can cause cancellation even if the sender is correct. However, the subset containing only correct processors will never be cancelled.

At the first step of the coin flipping stage, each sender randomly chooses a binary value C_1 and encrypts this under its public-key, E_1. Single source Byzantine agreement is then run in the whole network on the encrypted value $E_1(C_1)$. This is the turning point. After this point, every processor releases all decryption keys it holds. The value of the coin of an uncancelled subset is found by decrypting the sender's encrypted value (and taking some default if this encrypted value is neither 0 nor 1). Because every uncancelled subset contains at least one correct processor who has not protested, it must be the case that the coin of every uncancelled sender is decrypted after the turning point.

Again, this coin flipping protocol is built around the notion of a turning point. Most of the work is done before the turning point. At the turning point the coin is already determined, but its value is unpredictable to the adversary. After the turning point, the coin is revealed in one round, simply by having all (correct) processors broadcast all the information they hold. No Byzantine agreement is needed here—the protocol guarantees consistency of the values computed locally by the different processors.

In their paper, [ABCGM85] rigorously prove that the coin of the pure subset is unpredictable to the adversary before the turning point. Since the various actions of any correct processor in different subsets are independent, this argument can be used to show that the bit of the pure subset is "sufficiently independent" of other subsets' bits to give a subpolynomially biased coin. A similar argument can be made to show that global coins of different good committees are sufficiently independent, even if produced concurrently. Thus, this global coin flipping procedure can be used to implement Bracha's protocol in the presence of an adversary that can eavesdrop and rush, but is computationally bounded.

The property that the coin can be prepared in advance and then released in "one shot" actually enables the time-shared preparation of many independent coins, which can then be released one per round. Thus preprocessing can substantially decrease per coin preparation time. This idea was used in Ben-Or's asynchronous version of Bracha's protocol [Be85]. Taking it one step further, Feldman and Micali [FM85] eliminated the need to prepare many independent coins in advance. Instead, they proposed the use of a coin that is based on a cryptographically strong pseudorandom sequence generator (see Appendix II). As in [ABCGM85], each committee is again divided into subsets, and each subset has a unique sender and a set of inspectors. The basic idea is that in each subset the sender chooses a trapdoor permutation g and encrypts the trapdoor information associated with g in the public keys of each of the inspectors. Single source Byzantine agreement is then run throughout the committee on g, a random seed s in the domain of g, and the encryptions of the trapdoor information. Inspectors again have the opportunity to protest. In a good committee, every subset contains at least one correct processor, thus if no processor protests at least one processor in the subset will be able to compute the sequence $s, g^{-1}(s), g^{-2}(s), \ldots, g^{-i}(s) \ldots$. The pseudorandom bits to be used as the subset coins are obtained from this sequence. This method enables every processor in the subset (of which there is at least a single correct one) to send the subset coins (one at a time) and to convince all processors of their validity, without necessitating expensive agreements. The preprocessing time required to set up the generators is proportional to the size of the committee because deterministic single source Byzantine agreement is used as a subroutine.

4.6. Polynomially Biased Global Coins

In Sections 4.3 and 4.5 we discussed various protocols for achieving a global coin without bias or with a subpolynomially small bias. In particular, two such protocols were described that require communication exponential in the size of the subset to which the coin is global. As explained, this complexity is acceptable for implementing Bracha's protocol with committees of size $O(\log n)$, but prohibitive for larger committee sizes. In this section we describe a result of Broder [B85] that achieves an approximate solution to the global coin flipping problem. Informally, given a desired bias ε, Broder's protocol requires communication at most polynomial in ε^{-1} and runs in time $O(n)$. Since Bracha's protocol can tolerate a small bias in the coins of good committees, this performance makes it a potential candidate for implementing Bracha's protocol for larger committee sizes. (As usual, small bias is only a necessary condition, and the issue of independence should also be considered.)

Broder's protocol assumes the existence of a trapdoor function. It tolerates $t < n/3$ faults without authentication, $t < n/2$ faults otherwise. The protocol is based on the idea of taking as the value of the global coin the majority of binary values that are themselves the exclusive-or of individual coins. The structure of Broder's protocol closely resembles the structure of the protocol of Broder and Dolev [BD84], but circumvents the need to have a scheme where it is possible to verify, given $E(x)$ and y, whether $x = y$. Instead, Broder proposes that each processor p distribute the pieces of the secret coin using probabilistic encryption. Verification that p correctly encrypted its pieces is achieved (with high probability) as follows. Each processor p actually encrypts pieces of many different coins $c_1, c_2, \ldots, c_m$. Each other processor is allowed to request r of these coins to be revealed (all shares decrypted), for a suitable choice of r. If p is caught deviating from the protocol of any of these coins, then p is "erased," and ignored for the rest of the protocol. If no fault is found, the probability that *many* undisclosed c_i were improper is small. This now is the turning point. After it, all (correct) processors broadcast all the information they hold. The undisclosed coins are renumbered, from 1 to $m - nr$. Undisclosed coins $c'_1, c'_2, \ldots, c'_{m-nr}$ are recovered from their pieces. A default value is taken if inconsistent pieces were found. This might be under the control of the faulty processors, but the disclosure procedure

ensures that any processor publishing a large number of coins with inconsistent pieces will be caught. Thus the probability that many inconsistent pieces will be found is low. The coin G_i is defined to be the exclusive-or of all c_i' belonging to processors that were not erased. The majority of all G_i ($i = 1, 2, \ldots, n - mr$) is the final coin value.

If the c_i's are sufficiently uncorrelated, then the adversary can bias the outcome only by making some of the coins of faulty processors have inconsistent pieces. The bounds calculated by Broder require m to be $\Theta(n^2 \log^2(t/\varepsilon)/\varepsilon^2]$ in order to guarantee bias $\leqslant \varepsilon$. Because all the agreements can be done concurrently, the number of rounds is only $O(t)$. It is thus possible to make m large enough that $\varepsilon < 1/N$ at only polynomial (in N) message complexity.

We remark that Broder's protocol was not proved correct by a simulation argument, and he did not address the question of independence. However, we believe a simulation argument can be constructed and that the scheme affords the necessary independence. Assuming this correctness, we now examine the applications to Bracha's protocol.

When run in the context of Bracha's agreement protocol, Broder's work yields a global coin protocol for the committee requiring communication polynomial in the number of committees, and time linear in their size. This global coin protocol could thus be used for the nonconstructive Bracha scheme, with committees of size logarithmic in n. More importantly, when combined with Alon's explicit construction of a Bracha assignment in which the committees have size $\sqrt{n}$, Broder's scheme yields an agreement protocol with expected running time $O(\sqrt{n})$ and requiring message complexity polynomial in n. To our knowledge, in the model without a trusted dealer, no faster explicit agreement algorithm exists for $t = \Theta(n)$.

A simpler scheme, also based on majority with small number of potentially biased inputs, was proposed by Awerbuch, Blum, Chor, Goldwasser, and Micali [ABCGM85]. Although their solution is much more straightforward than that of Border, it requires a number of rounds that is too large to be useful in implementing Bracha's protocol. For this reason we omit further details.

4.7. Independence and Verifiable Secret Sharing

The problem of achieving independence in a Byzantine environment has already been raised as an important issue in coin flipping

protocols. Several attempts have been made to deal with this question in full generality, with mixed results ([CGMA85], [AGY86], [FM86]). The first to conceptualize the problem were Chor, Goldwasser, Micali, and Awerbuch [CGMA85]. They conceived of a scheme for "verifiable" secret sharing, in which each processor breaks its secret value into shares and then "proves" to each other processor that the share it receives is a valid one. Unfortunately, their protocol requires communication and computation exponential in the resiliency t (but polynomial in the security parameter). A key part in the protocol of [CGMA85] requires that each processor p prove to every other processor that it "knows what it is talking about," when it sends an encrypted value (that is, that p can print the value), in such a way that the proof reveals nothing about the value itself. This can be done, for example, by having processor p supply every other processor with a *zero-knowledge* proof [GMR85] that is (p) can decrypt its encryption key E_p. Intuitively, such proofs reveal no information other than the fact that p holds D_p. More efficient schemes, requiring communication and computation that are only polynomial in n were proposed in [AGY86] and [FM86], based on the intractability assumptions of deciding quadratic residuosity and factoring, respectively. A general, efficient, and conceptually simple scheme, based on any trapdoor function, was presented by Goldreich, Micali, and Wigderson [GMW86], using their result that every language in NP has a zero-knowledged proof. This result is directly applicable to the problem of flipping a global coin, and can be used to get a subpolynomially biased coin in the presence of a computationally bounded, eavesdropping adversary. However, their results cannot be used to generate the coins inside the committees in Bracha's protocol because these zero-knowledge proofs must be carried out serially, one prover at a time. This is because if proofs are carried out concurrently and the adversary can rush, then a faulty processor might use correct processors as oracles for providing proofs to statements it cannot prove by itself. Recently, Chor and Rabin [CR87] introduced a method for interleaving and alternating the various executions of verifiable secret sharing. Their method for achieving independence takes only $O(\log n)$ sequential executions, while requiring the same assumptions and complexity as in [GMW86]. These ideas can be used to implement the global coin on the ($\log n$ or larger size) committees of Bracha's protocol.

5. LOWER BOUNDS

In this section we discuss the few known lower bounds on randomized protocols for Byzantine agreement and distributed coin tossing. We begin by examining protocols that are required to produce a consistent decision upon termination (i.e., correct processors never reach disagreement).

Some lower bounds on resiliency carry over from the deterministic case to randomized protocols by simply fixing a particular sequence of coin tosses. Thus, for asynchronous Byzantine agreement in the unauthenticated model $t < n/3$ is essential [PSL80]. For asynchronous systems t must satisfy $t < n/3$ for both the authenticated and unauthenticated Byzantine cases, while $t < n/2$ is required for the failstop and omission models [BT83]. In both cases, for larger values of t, the adversary can simulate a partitioning of the network and thus prevent agreement.

Next, we examine lower bounds on the probability that a randomized agreement protocol will not terminate in k rounds, where $k \leqslant t$ [KY84, CMS85]. These bounds are obtained by modifying the lower bound argument of Dolev and Strong [DS82] for deterministic protocols. Let q_k denote the maximum probability, over all (nonadaptive) adversarial strategies, that the protocol does not terminate in k rounds ($k \leqslant t$). To prove a lower bound on q_k, one constructs a chain of scenarios (partial specifications of executions) with the following properties: each pair of adjacent scenarios is indistinguishable to some processor correct in both; the initial values force the decision in the first scenario to be 0; and a decision of 1 is forced in the last scenario. By attaching to each scenario the random strings corresponding to coin-flips, one gets a complete specification of an execution. If the length of the execution chain is at most m_k, then q_k must be at least m_k^{-1}, as otherwise the probability of not terminating in any of the scenarios in the chain is less than 1. In this case there must exist some particular random string causing termination in each of the k-round executions obtained by attaching this string to the scenarios in the chain. The properties of the chain now force a contradiction. As mentioned earlier, Chor, Merritt, and Shmoys have shown that for any $k \leqslant t$ the chain of k-round scenarios given by the Dolev and Strong proof of the $t + 1$ round lower bound for deterministic

agreement contains at most $m_k = 2(2\lceil n/\lfloor t/k \rfloor\rceil)^k$ scenarios. This implies $q_k \geqslant \frac{1}{2}(2\lceil n/\lfloor t/k \rfloor\rceil)^{-k}$ [KY84, CMS85].

In addition to examining the probability of not terminating in fewer than t rounds, Karlin and Yao [KY84] also explored how well randomized protocols that do not use authentication can do when $t > n/3$. As we mentioned earlier, such protocols must sometimes violate one of the two correctness conditions of Byzantine agreement. Karlin and Yao quantified how often such an error will occur. Specifically, they showed that for any randomized synchronous Byzantine agreement protocol, if no authentication mechanism is available and $t > n/3$, then there is an adversary that causes the protocol to fail with probability at least 1/3. Their argument is similar to arguments for upper bounds on t appearing in the literature on deterministic protocols.

In 1986 Cleve [Cl86], showed that for any 1-resilient r round coin flipping protocol for two processors there is an adversary strategy which causes a bias of at least $\Omega(1/r)$. Interestingly, the strategy employed by the faulty processor is merely to stop participating at a certain point in the protocol. The result holds even if the adversary is computationally bounded. By reducing the general case to the two processor case, Cleve showed that in any system of n processors there is a set of $\lceil n/2 \rceil$ processors that, if faulty, could bias the output by $\Omega[1/poly(n, r)]$, where $poly(n, r)$ is at most $O(n^3 r)$. Cleve also gives concise formal definitions of a general model for coin-flipping protocols.

Ben-Or and Linial [BL85] measured the influence of faulty processors in flipping a global coin by modeling global coin flipping as an n-person game in which each player has a source of unbiased coins. They consider the restrictive case of multistage games. These consist of several rounds. In each round every good processor flips an unbiased coin, the results are all made public, and then the faulty processors choose their values. Thus coins of faulty processors may depend on current round coins of correct processors, but not on future tosses. Finally, the global flip is defined by an arbitrary function of all the individual values. In this model, Ben-Or and Linial show that for any k, $1 \leqslant k \leqslant n$, there will always be some set of k players that together can bias the coin by at least $\Omega(k/n)$. This result can be interpreted, in our terminology, as implying that given a static, computationally unbounded adversary that can eavesdrop and rush messages, any global coin flipping protocol will have at least a constant positive bias when t is linear in n.

6. OPEN PROBLEMS

We now mention some specific open questions.

A. Better upper bounds.
   1. Is there a (cryptographic or noncryptographic, constructive or nonconstructive, tolerant or intolerant of eavesdropping and rushing) linearly resilient agreement protocol that runs in $O(1)$ expected time and tolerates Byzantine failures?
   2. Constructive agreement: The fastest linearly resilient agreement protocol known to us that is both constructive and requires neither a trusted dealer nor preprocessing is obtained by combining Bracha's protocol with the explicit construction of Alon. Its expected running time is $O(\sqrt{n})$. Are there faster constructive protocols tolerating $t = \Omega(n)$ faults? In particular, an affirmative answer would follow from an explicit construction of a Bracha assignment with committees of size $o(\sqrt{n})$.
   3. Computationally unbounded adversary with eavesdropping: The fastest linearly resilient agreement protocol that tolerates an eavesdropping and computationally unbounded adversary takes $O(t/\log n)$ expected number of rounds (synchronous model, weakly dynamic adversary) [CC84]. By contrast, in the case of a computationally bounded adversary, the fastest linearly resilient agreement protocols require $O(\log \log n)$ expected time without eavesdropping [DSS86], or $O(\log n)$ with eavesdropping [Br84]. This leaves a large gap between the expected running times for linearly resilient agreement protocols in the computationally bounded and the computationally unbounded models.

B. Lower bounds: No nontrivial lower bounds for the expected time to reach agreement are known. However, one should note that a linearly resilient cryptographic solution is known that, after preprocessing, requires $O(1)$ expected time per agreement [FM85].

C. Cryptographic coin flip: Design a linear time, linearly resilient global coin flipping protocol tolerating a strongly dynamic, computationally bounded, eavesdropping adversary, having the following properties: It should have subpolynomially small bias, coins of good committees should be (almost)

independent, and coins of bad committees (almost) independent of coins of good committees. This last property is needed in the Bracha-like construction of Dwork, Shmoys, and Stockmeyer.

D. Noncryptographic coin flip: Consider the model with a computationally unbounded adversary who can eavesdrop and rush, and t is linear in n. Either design a global coin flipping protocol with constant bias, or prove that such a protocol does not exist. (The lower bound of [BL85] only implies that the bias is at least a constant $\neq 0$.) For multiround protocols, the question is open for all variants of adaptivity (static, weakly dynamic, and strongly dynamic).

NOTE

Feldman and Micali ["Optimal algorithms for Byzantine Agreement," *Proc. 17th ACM Symp. Theory of Computing* 148–161, 1988] announced a linearly resilient Byzantine agreement protocol that runs in constant expected time.

APPENDIX I

This appendix contains a chronologically ordered list of principal randomized algorithms for Byzantine agreement. The salient points of each algorithm are mentioned briefly.

Ben-Or 1983 "Another advantage of free choice: Completely asynchronous agreement protocols." Resiliency: $t < n/5$; expected running time: $2^{\theta(n)}$ if $t = \Theta(n)$; $O(1)$ with resiliency $t = O(\sqrt{n})$; adversary: strong dynamic, intrusive.

Bracha, Toueg 1983: "Resilient consensus protocols." Resiliency: fail-stop $t < n/2$, unauthenticated Byzantine $t < n/3$; expected running time: $2^{\theta(n)}$ when $t = \Theta(n)$; $O(1)$ at resiliency $t = O(\sqrt{n})$; assumptions: probabilistic message delays; notes: probabilistic analysis based on behavior of message system; processors themselves are deterministic.

Rabin 1983 "Randomized Byzantine generals." Model: unauthenticated Byzantine processors, unforgeable trusted dealer; resiliency: $t < n/4$ (synchronous) or $t < n/10$ (asynchronous); expected running time: $O(1)$; Trusted dealer precomputes secret random bits and distributes shares to all participants. If coin is bad (different players see different values) agreement may be delayed but disagreement never occurs. Assumptions: unforgeable trusted dealer; adversary: strongly dynamic.

Bracha 1984: "An asynchronous $\lfloor \frac{n-1}{3} \rfloor$-resilient consensus protocol." Model: unauthenticated Byzantine processors; resiliency: $t < n/3$; expected running time: $2^{\theta(n)}$ when $t = \Omega(n)$, $O(1)$ at resiliency $t = O(\sqrt{n})$; adversary: strongly dynamic.

Toueg 1984: "Randomized Byzantine agreements." Model: authenticated Byzantine processors, authenticated trusted dealer; resiliency: $t < n/2$ synchronously, $t < n/3$ asynchronously; expected running time: $O(1)$; notes: trusted dealer precomputes secret random bits and distributes shares to all participants. Assumptions and adversary: as in Rabin 1983.

Chor, Coan 1984: "A simple and efficient randomized Byzantine generals protocol." Model: unauthenticated Byzantine processors, synchronous; resiliency: $t < n/3$; expected running time: $O(t/\log n)$; adversary: weakly dynamic, computationally unbounded, rushing and eavesdropping allowed.

Bracha 1985: "An $O(\log n)$ expected rounds randomized Byzantine generals protocol." Model: Byzantine processors, synchronous communication; resiliency: $t < n/(3 + \varepsilon)$ without authentication, $t < n/(2 + \varepsilon)$ with authentication; expected running time: $O(\log n)$ as published, but can be improved to $O[\log n/\log (n/t)]$; assumptions: processors assigned to committees according to Bracha assignment; comments: inherits the adversary properties of the coin flipping protocol used by the individual committees, provided independence is guaranteed among concurrent executions of this protocol by different good committees; adversary: computationally unbounded, rushing without eavesdropping; or computationally bounded, rushing with eavesdropping (assuming trapdoor functions exist).

Ben-Or 1985: "Fast asynchronous Byzantine agreement." Model: unauthenticated Byzantine processors, asynchronous communication; expected running time: $O[\log n(\log \log n)^c]$; assumptions: processors assigned to committees according to a Bracha assignment, recursively (three times); resiliency: $t < n/7$; adversary: rushing allowed, eavesdropping forbidden. Comments: inherits the adversary properties of the coin flipping protocol used by the individual committees, as above.

Chor, Merritt, Shmoys 1985: "Simple constant-time consensus protocols in realistic failure models." Model: omission faulty processors; resiliency: $t < n/4$ synchronously, $t < n/6$ asynchronously (recently improved to $t < n/2$ and $t < n/3$, respectively); expected running time $O(1)$; assumptions: for some adversaries the

existence of trapdoor functions; adversaries: weakly dynamic and computationally unbounded or strongly dynamic but computationally bounded (assuming trapdoor functions exist).

Feldman, Micali 1985: "Byzantine agreement in constant expected time." Model: Byzantine processors, synchronous communication; polynomially bounded, weakly dynamic adversary (requires intractability assumptions); resiliency: $t < n/(2 + \varepsilon)$ if authentication is available, $t < n/(3 + \varepsilon)$ if otherwise; expected running time: $O(1)$ per agreement; comments: requires $\Omega(\log n)$ preprocessing if starting with a Bracha assignment, at least $\Omega(n)$ preprocessing otherwise.

Dwork, Shmoys, Stockmeyer 1986: "Flipping persuasively in constant expected time." Model: Byzantine processors, synchronous communication; tolerates rushing without eavesdropping with computationally unbounded adversary; tolerates blocking with computationally bounded adversary (assuming trapdoor functions exist); resiliency: $t = O(n/\log n)$; expected running time $O(1)$, needs no preprocessing; comments: can be bootstrapped via a Bracha-like construction to yield resiliency $t < n/(4 + \varepsilon)$ and $O(\log \log n)$ expected time agreement algorithm in the model in which the adversary can rush but cannot eavesdrop.

APPENDIX II

In this appendix we briefly describe some of the cryptographic techniques used as subroutines in various randomized agreement protocols. The descriptions are generally kept short and on the intuitive level. Pointers to the original references are given. (In addition, the interested reader may find an extensive survey of modern cryptology in [Ri89].)

One-Way Functions

One-way functions [DH76] are functions that are easy to evaluate but hard to invert. Given an x, it is easy to compute $y = f(x)$, but for most y it is infeasible to find any z such that $y = f(z)$.

Trapdoor Functions

Trapdoor functions [DH76] are one-way functions that, given additional "trapdoor information," are easy to invert. Trapdoor

permutations (one-to-one functions) are the basis for public-key encryptions. The user generates an instance of the function f and publishes it (this is the public encryption key), while keeping the trapdoor information secret (this is the private decryption key).

As a concrete example, we describe the RSA public-key scheme [RSA78]. The public key consists of a pair (e, N) where $N = p \cdot q$ is the product of two large primes, and e is relatively prime to $(p-1) \cdot (q-1)$. The one-way permutation is $f(x) = x^e \pmod{N}$. The secret trapdoor information is the factorization of N, which enables the legitimate user to efficiently extract eth roots $f^{-1}(y) = y^{1/e} \pmod{N}$.

Probabilistic Encryption

In a deterministic encryption scheme every message x has a unique encryption y. By contrast, in a probabilistic scheme [GM82] every message x is encrypted by a y chosen at random from a large set of elements satisfying a relation $R(x, y)$. The remarkable property of probabilistic public-key schemes is the ability to prove very strong notions of statistical security, based on concrete complexity theoretic assumptions.

As an example, consider the following scheme [GM82], which is based on the intractability assumption of deciding quadratic residuosity modulo composite numbers. Let $N = p \cdot q$ be as in the RSA public-key encryption scheme, described above. To encrypt the bit 0, a square modulo N is chosen at random among all $N/4$ squares. To encrypt the bit 1, a nonsquare with Jacobi symbol 1 modulo N is chosen at random among all $N/4$ such nonsquares. (Longer messages are encrypted by concatenating the encryptions of individual bits.) In both cases, it is possible to perform such a choice efficiently. Given the factorization of N, it is possible to efficiently distinguish squares from nonsquares, and therefore to decrypt. Other efficient probabilistic public-key schemes, based on the intractability of factoring or inverting RSA appear in [ACGS84, BG85].

Authentication

An authentication mechanism enables a user A to send unforgeable messages to user B so that, when relayed to a third party C, the message is recognized by C as one originating from A. One possible mechanism for authentication uses "digital signatures" based on

trapdoor onto functions [DH76]: to authenticate a message y, user A "signs" it by publishing $x = f_A^{-1}(y)$, where f_A is user A's trapdoor function that only A can invert. Thus, without the secret trapdoor information, it is hard to forge messages (compute f_A^{-1}), but it is easy to verify that a message is properly signed (compute f_A).

Probabilistic methods for digital signatures have also been proposed [GMY83, GMR84]. In particular, [GMR84] presents a scheme for which, under the assumption that factoring is intractable, it is provably hard for a computationally bounded adversary to forge any additional message even if first supplied with the legitimate signatures of polynomially many messages chosen by the adversary. This is a much stronger requirement that just infeasibility of forging. Such a scheme is particularly valuable in Byzantine agreement protocol in which processors are required to sign any message they have received in previous rounds (such as the protocol of [PSL80]).

Cryptographically Secure Pseudorandom Sequence Generators

These are deterministic algorithms that, when given as input a short random sequence of k bits called the *seed*, generate a longer, pseudorandom sequence of k^c bits (where $c \geqslant 1$ is any constant). The set of 2^k pseudorandom sequences cannot be distinguished by any polynomial-time statistical test (polynomial in k) from the much larger set of all 2^{k^c} sequences of length k^c [BM84, Y82]. In particular, if the seed is hidden, then the next bit in the output sequence cannot be predicted by any probabilistic polynomial time protocol with probability polynomially better than 1/2, given any initial prefix of the sequence.

Under suitable complexity theoretic assumptions, it is possible to design provably secure generators of this type. Efficient generators were designed assuming the intractability of computing discrete logarithms modulo a prime [BM84], of deciding quadratic residuosity modulo a composite number [BBS82], and of factoring or inverting RSA [ACG84]. Yao [Y82] and Levin [L85] have made progress towards showing that the existence of one-way functions is a necessary and sufficient condition for generating pseudorandom sequences.

Secret Sharing

This is a method for a sender (the "dealer") to split a secret s into n pieces, in such a way that no collection of t pieces gives any

information about the secret s, but any collection of $t + 1$ pieces can be used to efficiently reconstruct the secret. We describe the following method for secret sharing [Sh79], assuming p is a prime satisfying $p \geqslant n$, and the secret s is an integer $0 \leqslant s \leqslant p - 1$. The dealer chooses a random polynomial $P(x)$ of degree t such that $P(0) = s$. The piece of the secret sent by the dealer to each player i is the value of the polynomial at the point i, $P(i)$, where $1 \leqslant i \leqslant n$. We remark that the correctness of this scheme is not based on any unproved complexity theoretic assumptions.

ACKNOWLEDGMENTS

The authors are indebted to Danny Dolev, Ron Fagin, Oded Goldreich, Jim Hafner, Joe Halpern, Silvio Micali, Larry Stockmeyer, and Ray Strong for their comments and suggestions on earlier versions of this paper. Research supported in part by a Bantrell Postdoctoral Fellowship.

REFERENCES

[ACGS84] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, "RSA and Rabin Functions: Certain Parts Are as Hard as the Whole," *SIAM J. Comput.* 17(2): 194–209 (1988).

[A85] N. Alon, private communication.

[AGY86] N. Alon, Z. Galil, and M. Yung, private communication.

[ABCGM85] B. Awerbuch, M. Blum, B. Chor, S. Goldwasser, and S. Micali, "How to implement Bracha's $O(\log n)$ Byzantine agreement protocol." unpublished manuscript.

[B85] A. Z. Broder, "A provably secure polynomial approximation scheme for the distributed lottery problem," *Proc. 4th Annu. ACM Symp. Principles Distributed Computing* 136–148 (1985).

[BD84] A. Z. Broder and D. Dolev, "Flipping coins in many pockets," *Proc. 25th Symp. Foundations Computer Science* 157–170 (1984).

[Be83] M. Ben-Or, "Another advantage of free choice: Completely asynchronous agreement protocols," *Proc. 2nd Annual ACM Symp. Principles Distributed Computing* 27–30 (1983).

[Be85] M. Ben-Or, "Fast asynchronous Byzantine agreement," *Proc. 4th Annu. ACM Symp. Principles Distributed Computing* 149–151 (1985).

[BL85] M. Ben-Or and N. Linial, "Collective coin flipping, robust voting games, and minima of Banzhaf values", *Proc. 26th Annu. Symp. Foundations Computer Science* 408–416 (1985).

[BBS82] L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudo-random number generator," *SIAM J. Comput.* 15: 364–383 (1986).

[BM84] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. Comput.* 13(4): 850–864 (1984).

[Br84] G. Bracha, "An asynchronous $(n - 1)/3$-resilient consensus protocol," *Info. and Comput.* 75(2): 130–144 (1987).

[Br85] G. Bracha, "An $O(\log n)$ expected rounds randomized Byzantine generals protocol," *JACM* 34(4): 910–920 (1987).

[BT83] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *JACM* 32(4): 824–840 (1985).

[C85] B. Chor, "Critical remarks on the Broder–Dolev coin flipping protocol," unpublished notes.

[CC84] B. Chor and B. Coan, "A simple and efficient randomized Byzantine agreement protocol," *IEEE Transact. Software Engineering* SE-11(6): 531–539 (1985).

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," *Proc. 26th IEEE Conference Foundations Computer Science* 383–395 (1985).

[Cl86] R. Cleve, "The impossibility of secure coin flips when half the process are faulty," *Proc. 18th Annu. ACM Symp. Theory Computing* 364–369 (1986).

[CMS85] B. Chor, M. Merritt, and D. Shmoys, "Simple constant-time consensus protocols in realistic failure models," *Proc. 4th Annu. ACM Symp. Principles Distributed Computing* 152–162 (1985).

[CR87] B. Chor and M. O. Rabin, "Achieving independence in logarithmic number of rounds," *Proc. 6th Annu. ACM Symp. Principles Distributed Computing* 260–268 (1987).

[DLM82] R. DeMillo, N. Lynch and M. Merritt, "Cryptographic protocols," *Proc. 14th Annu. ACM Symp. Theory Computing* 383–400 (1982).

[DH76] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transact. Information Theory* IT-22: 644–654 (1986).

[DS82] D. Dolev and H. R. Strong, "Polynomial protocols for multiple processor agreement," *Proc. 14th Annu. ACM Symp. Theory Computing* 401–407 (1982).

[DSS86] C. Dwork, D. Shmoys, and L. Stockmeyer, "Flipping persuasively in constant expected time," *Proc. 27th Symp. Foundations Computer Sci.* 222–232 (1986).

[F83] M. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," Technical Report, Department of Computer Science, Yale University, 1983.

[FLa82] M. Fischer and L. Lamport, "Byzantine generals and transaction commit protocols," SRI Technical Report Op. 62.

[FLP83] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *JACM* 32(2): 374–382 (1985).

[FLy82] M. Fischer and N. Lynch, "A lower bound for the time to assure interactive consistency," *Inf. Process. Lett.* 14(4): 183–186 (1982).

[FM85] P. Feldman and S. Micali, "Byzantine agreement in constant expected time (and trusting no one)," *Proc. 26th Annu. Symp. Foundations Computer Sci.* 267–276 (1985).

[FM86] P. Feldman and S. Micali, private communication.

[GM82] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comp. System Sci.* 28(2): 270–299 (1984).

[GMR84] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.* 17(2): 281–308 (1988).

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.* 18(1): 186–208 (1988).

[GMY83] S. Goldwasser, S. Micali, and A. Yao, "Strong signature schemes," *Proc. 23rd Annu. Symp. Foundations Computer Sci.* 431–439 (1983).

[GMW86] O. Goldreich, S, Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design," *Proc. 27th Symp. Foundations Computer Sci.* 174–187 (1986).

[H83] V. Hadzilacos, "Byzantine agreement under restricted types of failures (not telling the truth is different from telling lies)," *Harvard University Technical Report TR-19-83*, Department of Computer Science, 1983.

[Ha85] J. Hastad, "Solving simultaneous modular equations of low degree," *SIAM J. Comput.* 17(2): 336–341 (1988).

[KY84] A. Karlin and A. Yao, "Probabilistic lower bounds for Byzantine agreement," unpublished manuscript.

[L85] L. A. Levin, "One way functions and pseudo random generators," *Proc. 17th Annu. ACM Symp. Theory Computing* 363–365 (1985).

[LFF82] N. Lynch, M. Fischer, and R. Fowler, "A simple and efficient Byzantine generals protocol," *Proc. 2nd Symp. Reliability Distributed Software Database Systems* 46–52 (1982).

[LSP82] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM TOPLAS* 4(3): 382–401 (1982).

[PSL80] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *JACM* 27(2): 228–234 (1980).

[R83] M. Rabin, "Randomized Byzantine generals," *Proc. 24th Annu. Symp. Foundations Computer Sci.* 403–409 (1983).

[Re85] R. Reischuk, "A new solution for the Byzantine generals problem," *Inf. Control 64* No. 1–3: 23–42 (1985).

[Ri89] R. L. Rivest, "Cryptology," In *Handbook of Theoretical Computer Science* (M. Nivat, ed.). In press.

[RSA78] R. L. Rivest, A. Shamir, and L. Adelman, "A method for obtaining digital signatures and public key cryptosystems," *Commun. ACM* 21: 120–126 (1978).

[Sh79] A. Shamir, "How to share a secret," *Commun. ACM* 22(11): 612–613 (1979).

[ST84] T. K. Srikanth and S. Toueg, "Simulating authenticated broadcasts to derive simple fault-tolerant protocols," *TR 84-623*, Department of Computer Science, Cornell University.

[ST85] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *Proc. 4th Annu. ACM Symp. Principles Distributed Computing* 71–86 (1985).

[T84] S. Toueg, "Randomized Byzantine agreements," *Proc. 3rd Annu. ACM Symp. Principles Distributed Computing* 163–178 (1984).

[TC84] R. Turpin and B. Coan, "Extending binary Byzantine agreement to multivalued Byzantine agreement," *Inf. Process. Lett.* 18(2): 73–76 (1984).

[TPS85] S. Toueg, K. Perry, and T. K. Srikanth, "Fast distributed agreement," *Proc. 4th Annu. ACM Symp. Principles Distributed Computing* 87–101 (1985).

[Y82] A. C. Yao, "Theory and applications of trapdoor functions," *Proc. 23rd IEEE Symp. Foundation Computer Sci.* 80–91 (1982).

[Y84] A. C. Yao, public lecture.

BIASED COINS AND RANDOMIZED ALGORITHMS

N. Alon and M. O. Rabin

ABSTRACT

A slightly random source is a source of bits, where the bias of each bit, between $1/2 + \varepsilon$ and $1/2 - \varepsilon$ for some $\varepsilon > 0$, is fixed by an adversary who has a complete knowledge of all the previous bits. We study the properties of sequences of n consecutive bits generated by such a source. In particular we show that for most subsets S of half of the n-binary vectors, even a fixed bias $\varepsilon > 0$, and arbitrarily large, n will not enable the adversary (who knows S) to avoid it with probability approaching 1 as n tends to infinity. Also, for every n and every $S \subseteq \{0,1\}$, $|S| = 2^{n-1}$, if $\varepsilon < 1/(2\sqrt{n})$ then the adversary cannot decrease the probability of landing in S below 1/6. These results mean that for randomized algorithms such as primality testing, even a fairly biased coin will produce good answers, without any change in the algorithm.

Advances in Computing Research, Volume 5, pages 499–507.

ISBN: 0-89232-896-7

1. INTRODUCTION

Several applications, such as randomized algorithms [Ra], require a source of fair coin flips. The available physical sources are imperfect. The simplest model of such an imperfect source of random bits is a coin whose flips are independent, and each has a fixed (and unknown) bias. von Neumann [vN] gave a simple algorithm for generating absolutely random independent bits from such a coin. Blum [Bl] (see also Elias [El]) generalized this algorithm to the case where the imperfect random source is an n-state Markov chain. This algorithm, however, is not useful for very large n since it produces bits only when states are repeated. A very general model of an imperfect source of randomness is considered by Santha and Vazirani [SV] and by Vazirani [V] (see also [CG, VV]). In this model, the next bit is an output of a coin whose bias (between $1/2 + \varepsilon$ and $1/2 - \varepsilon$ for some $0 < \varepsilon < 1/2$) is fixed by an adversary who has a complete knowledge of all the previous bits. Thus the previous bits can condition the next bit in an arbitrary bad way. Such a source is called a slightly random source in [SV], and as is explained in [Mu] it includes the known physical sources of randomness as, e.g., zener diodes. The algorithms of [SV, V], for extracting almost fair coin flips from such a model, use the existence of at least two *independent* slightly random sources. It is not clear at all that such an assumption is practical. The bad behavior of the sources might arise from the environment's influence and then the sources influence each other. On the other hand, it is trivial to show that no such algorithm that uses a single slightly random source exists. Thus it is interesting to check the properties of a single slightly random source. In this chapter we show that under reasonable assumptions n consecutive output bits of a single slightly random source form a "reasonable random" n-binary vector. In a typical randomized algorithm (such as the known primality test algorithms see [Ra]), we choose randomly an n vector and we succeed if this vector corresponds to a "witness". Suppose that the set of witnesses S forms a constant fraction c $(0 < c < 1)$ of all 2^n possible vectors. Our first observation is that if $\varepsilon(n) = d/\sqrt{n}$, then even an adversary who tries to avoid S and chooses the bias of every flip between $1/2 - \varepsilon(n)$ to $1/2 + \varepsilon(n)$ has a probability $f(c, d) > 0$ (independent of n) of getting an n vector in S. This result is sharp.

More surprising is our second result, which shows that under the (plausible) assumption that the set S of witnesses is a random set,

even fixed bias $\varepsilon > 0$ and arbitrarily large n will not enable the adversary (who knows S) to avoid it with probability $\to 1$ as $n \to \infty$. Thus, for example, if $\varepsilon = 0.05$ and n is large, then for almost every subset S of the set of all 2^n binary vectors, an adversary who knows S and tries to avoid it by choosing the bias of each of his coin flips between $1/2 - \varepsilon$ to $1/2 + \varepsilon$ (taking into account the results of the previous flips), will get a vector of S with probability $> 1/4$. Therefore, for almost all sets S, a weakly random source is reasonable, even under adversary assumptions.

Very recently, Vazirani and Vazirani [VV] (see also [CG] for some extensions) have found a clever algorithm that works for *every* set S of $c \cdot 2^n$ witnesses in the following sense. In the algorithm, a single slightly random source is used to produce a large polynomial number of n-vectors, at least one of which belongs to S with probability $f(c) > 0$ (independent of n).

In the present chapter we do not discuss possible algorithms to obtain witnesses with high probability, but rather study the properties of the bits produced directly by a single weakly random source. We believe that this supplies a better understanding of the behavior of such a source. Moreover, in our approach (unlike in the more sophisticated algorithms of [VV, CG]) we need only n slightly random bits to produce an n bit number, and we do not need any extra space.

The chapter is organized as follows. In Section 2 we find, for every bias $0 < \varepsilon < 1/2$, for every n, and for every $0 \leqslant k \leqslant 2^n$, the "worst possible" set S of n-vectors of cardinality k. In Section 3 we consider random sets S. Section 4 contains some concluding remarks.

2. THE EXTREMAL CASE

We begin with some notation. For $n \geqslant 1$ let $N = N(n)$ denote the set of all binary vectors of length n. For $0 \leqslant \varepsilon \leqslant 1/2$, let $F(n, \varepsilon)$ be the following set of strategies F for choosing a binary vector $(x_1, x_2, \ldots, x_n) \in N$. $x_1 \in \{0, 1\}$ is chosen according to the probability distribution $\text{Prob}(x_1 = 0) = \rho_1 = \rho_1(F)$ where $1/2 - \varepsilon \leqslant \rho_1 \leqslant 1/2 + \varepsilon$. (The value of ρ_1 is determined by the strategy F.) For every given binary values of $x_1, \ldots, x_{i-1}$, $x_i \in \{0, 1\}$ is chosen according to the probability distribution $\text{Prob}(x_i = 0) = \rho_i$, where $\rho_i = \rho_i(F, x_1, \ldots, x_{i-1})$ satisfies $1/2 - \varepsilon \leqslant \rho_i \leqslant 1/2 + \varepsilon$.

Let S be a set of binary vectors of length n. Define $P(S,\varepsilon) = \min_{F\in F(n,\varepsilon)}\text{Prob}\{(x_1,x_2,\ldots,x_n)\in S;\ (x_1,x_2,,\ldots,x_n)$ is chosen according to $F\}$. Thus $P(S,\varepsilon)$ is the minimal possible probability of a binary vector to be in S if it is chosen according to one of the strategies in $F(n,\varepsilon)$. [That is, according to biased coin flips, each in the range $(1/2-\varepsilon, 1/2+\varepsilon)$, where the bias is chosen by an adversary who knows the previous flips results, knows S, and tries to avoid it.]

Define a linear order on the set of all binary vectors of length n as follows: If $u = (u_1,u_2,\ldots,u_n)$, $v = (v_1,v_2,\ldots,v_n)$ then $u \leqslant v$ iff $\sum_{i=1}^n u_i < \sum_{i=1}^n v_i$ or $\sum_{i=1}^n u_i = \sum_{i=1}^n v_i$ and $\sum_{i=1}^n u_i 2^i < \sum_{i=1}^n v_i 2^i$. A set S of binary vectors is called *compressed* if $v\in S$ and $u \geqslant v \to u\in S$. It is easy to check that if S is compressed then it contains all vectors with at most j 0's and possibly some vectors with precisely $j+1$ 0's, where $0 \leqslant j < n$ satisfies

$$\sum_{i=0}^{j}\binom{n}{i} \leqslant |S| \leqslant \sum_{i=0}^{j+1}\binom{n}{i}. \tag{2.1}$$

Finally, for a set $S \subseteq N$ we denote by CS the unique compressed set of cardinality $|S|$.

PROPOSITION 2.1. (i) *For every* $0 \leqslant \varepsilon \leqslant 1/2$ *and for every set* S *of binary vectors*

$$P(S,\varepsilon) \geqslant P(CS,\varepsilon).$$

(ii) *Suppose* $0 \leqslant \varepsilon \leqslant 1/2$ *and* S, j *satisfy* (2.1). *Put* $r = |S| - \sum_{i=0}^{j}\binom{n}{i}$. *Then*

$$P(CS,\varepsilon) = \sum_{i=0}^{j}\binom{n}{i}(1/2+\varepsilon)^i(1/2-\varepsilon)^{n-i} + r\cdot(1/2+\varepsilon)^{j+1}(1/2-\varepsilon)^{n-j-1}.$$

That is, the best adversary's strategy to avoid CS is to bias each flip, as strongly as he can, toward 0.

Proof. The set N of all 2^n binary vectors can be naturally represented by the set of all leaves of a rooted binary tree of height n. Each left edge represents a zero and each right edge a 1. A leaf

corresponds to the vector arising from the edges of the unique path from the root to the leaf. Any strategy $F \in F(n, \varepsilon)$ is an assignment of a pair of probabilities $1/2 - \varepsilon \leqslant \rho,\ q \leqslant 1/2 + \varepsilon,\ \rho + q = 1$ to each pair of edges that emanates from a common parent. It is easy to check that we can assume that the adversary always chooses, for each pair of probabilities, either $\rho = 1/2 - \varepsilon$ or $\rho = 1/2 + \varepsilon$. Indeed, from each parent he will prefer to go with the highest possible probability to the child from whom he has more chances to avoid S. Thus, we can assume that each flip is as biased as possible.

For a vector $v = (v_1, v_2, \ldots, v_n) \in N$ put $\rho(v) = (1/2 - \varepsilon)^{|\{i: v_i = 1\}|} (1/2 + \varepsilon)^{|\{j: v_j = 0\}|}$. If each flip is as biased as possible then the sequence of probabilities of the leaves of our tree is clearly some permutation of the numbers $\{\rho(v) : v \in N\}$. The total probability of vectors in S is thus at least the sum of the $|S|$ smallest numbers in the sequence $\{\rho(v) : v \in N\}$. These numbers are, however, precisely those whose sum is given in part (ii) of the proposition, and if S is compressed the strategy of always preferring 0 achieves this bound. This completes the proof. □

Since a binomial distribution can be approximated by a normal one, one can get a very good estimate for the bound supplied by Proposition 2.1. Thus, for example, it implies that for all fixed c, $d > 0$ there exists an $f = f(c, d) > 0$ such that if $S \subseteq N$, $|S| = c \cdot 2^n$ and $\varepsilon = \varepsilon(n) = d/\sqrt{n}$ then $P(S, \varepsilon) \geqslant P(CS, \varepsilon) > f$. On the other hand if $\varepsilon = \varepsilon(n) = dg(n)/\sqrt{n}$ where $g(n) \to \infty$ arbitrarily slowly it is easy to check that $\lim_{n \to \infty} P[CS, \varepsilon(n)] = 0$.

As a special case we mention that if $|S| = 1/2 \cdot 2^n$, $\varepsilon = \varepsilon(n) = 1/(2\sqrt{n})$ then the normal approximation gives that $P(S, \varepsilon) \geqslant 1/6$.

REMARK 2.2. The assertion of Proposition 2.1 can be easily generalized to the case of a random "dice" ($t > 2$ possible results at each flip). This can be used to improve some of the results of [TRV]. We omit the details.

3. THE RANDOM CASE

In this section we show that for a random subset S of binary vectors of length n, even a fixed bias $\varepsilon > 0$ and arbitrarily large n will not enable the adversary, who knows S, to avoid it with probability $\to 1$ as $n \to \infty$.

Let S be a random subset of N. That is, each $v \in N$ is in S with probability $1/2$, independently. For each $0 \leqslant \varepsilon \leqslant 1/2$, $P(S, \varepsilon)$ is now a random variable (on the space of all possible 2^{2^n} subsets S). Let $E = E_{n,\varepsilon} = E[P(S, \varepsilon)]$ and $\sigma = \sigma_{n,\varepsilon} = \sigma[P(S, \varepsilon)]$ denote the expected value and the standard deviation of $P(S, \varepsilon)$.

THEOREM 3.1. *For every $\varepsilon > 0$ that satisfies $1/2 + 2\varepsilon^2 + 2\varepsilon < 1$ and for every n:*

$$E_{n,\varepsilon} \geqslant \frac{1}{2}\left(1 - \frac{\sqrt{2\varepsilon}}{1 - \sqrt{1/2 + 2\varepsilon^2 + 2\varepsilon}}\right)$$

and

$$\sigma_{n,\varepsilon} \leqslant 1/2(1/2 + 2\varepsilon^2 + 2\varepsilon)^{n/2}.$$

Thus, for example, if $\varepsilon = 0.05$ then $E_{n,\varepsilon} \geqslant 1/3$ and $\sigma_{n,\varepsilon} \leqslant (0.78)^n$. Hence, by Chebyshev's inequality [F, p. 219], for random $S \subseteq N(n)$, the probability that $P(S, \varepsilon)$ is smaller than 0.3 is at most $(0.03)^{-2} \cdot (0.78)^{2n}$. That is, almost for every S, $P(S, \varepsilon) \geqslant 0.3$.

To prove our theorem we need some preparations and a probabilistic lemma.

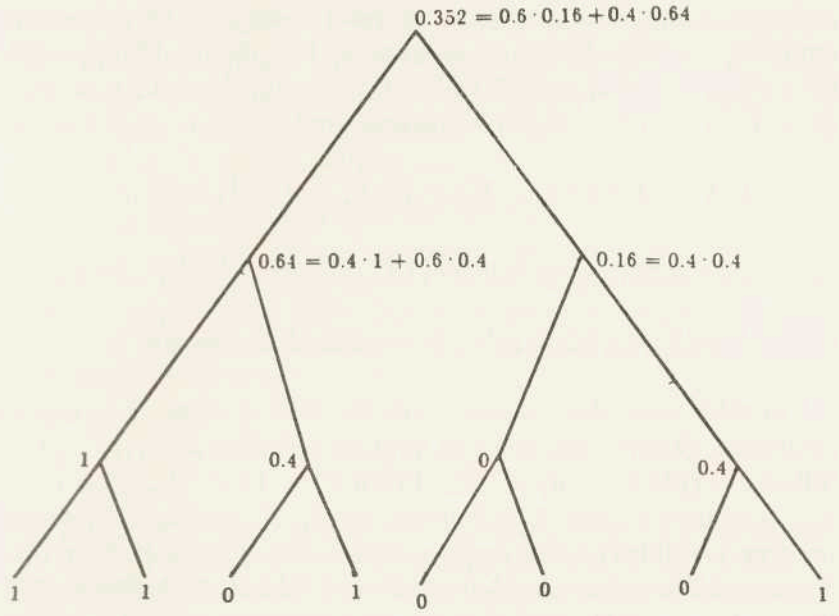
For a given set S and a given ε, one can easily convince himself that $P(S, \varepsilon)$ can be computed as follows: Let T be a binary tree of depth n whose leaves correspond naturally to the binary vectors of length n. Label a leaf corresponding to a vector v by 0 if $v \notin S$ and by 1 if $v \in S$. Now label, recursively, each parent f of the already labeled children s_1, s_2 with the following real number: Suppose s_i is labeled by r_i, then the label of f is $(1/2 + \varepsilon)\min(r_1, r_2) + (1/2 - \varepsilon)\max(r_1, r_2)$. One can check that the label of the root is $P(S, \varepsilon)$. In Figure 1 we have an example of $n = 3$, $S = \{000, 001, 011, 111\}$, $\varepsilon = 0.1$). Here $P(S, \varepsilon) = 0.352$.

Suppose now that S is a random set of vectors in $N(n)$. We have to estimate the expected value and the standard deviation of the random variable $P(S, \varepsilon)$. We need the following lemma.

LEMMA 3.2. *Let X_1, X_2 be two independent random variables, each with expected value E and standard deviation σ. Put*

$$Y = (1/2 + \varepsilon)\min(X_1, X_2) + (1/2 - \varepsilon)\max(X_1, X_2)$$

$$= \frac{X_1 + X_2}{2} - \varepsilon |X_1 - X_2|.$$



Then

$$E(Y) \geqslant E - \sqrt{2\varepsilon}\sigma \tag{3.1}$$

$$\sigma(Y) \leqslant \sqrt{1/2 + 2\varepsilon^2 + 2\varepsilon}\,\sigma. \tag{3.2}$$

Figure 1.

Proof. By Jensen's inequality $E(|X_1 - X_2|)^2 \leqslant (E|X_1 - X_2|)^2$. However, $E(|X_1 - X_2|)^2 = E[(X_1 - X_2)^2] - [E(X_1 - X_2)]^2 = \mathrm{Var}(X_1 - X_2) = 2\sigma^2$. Hence $E|X_1 - X_2| \leqslant \sqrt{2}\sigma$ and (3.1) follows. To prove (3.2) we compute $\sigma^2(Y) = \mathrm{Var}(Y) = E(Y^2) - [E(Y)]^2$.

$$\begin{aligned}\mathrm{Var}(Y) &= 1/4\,\mathrm{Var}(X_1 + X_2) + \varepsilon^2\,\mathrm{Var}|X_1 - X_2| \\ &\quad + \varepsilon\{E(X_1 + X_2)\,E|X_1 - X_2| - E[(X_1 + X_2)|X_1 - X_2|]\} \\ &\leqslant 1/2\sigma^2 + 2\varepsilon^2\sigma^2 + \varepsilon\{E(X_1 + X_2)\,E|X_1 - X_2| \\ &\quad - E[(X_1 + X_2)|X_1 - X_2|]\}.\end{aligned}$$

For every two random variables Z, T, $|E(Z)\,E(T) - E(ZT)| \leqslant \sqrt{\mathrm{Var}\,Z}\sqrt{\mathrm{Var}\,T}$. [This is the well-known fact that the correlation

constant is, in absolute value, at most 1, or can be derived by applying Cauchy–Schwarz inequality to obtain $\{E[(Z-EZ)(T-ET)]\}^2 \leqslant E(Z-EZ)^2 E(T-ET)^2$.] Applying this to $Z = X_1 + X_2$, $T = |X_1 - X_2|$ we conclude that

$$E(X_1+X_2)\,E|X_1-X_2| - E[(X_1+X_2)|X_1-X_2|]$$

$$\leqslant \sqrt{\mathrm{Var}(X_1+X_2)\cdot \mathrm{Var}|X_1-X_2|} \leqslant \sqrt{4\sigma^4} = 2\sigma^2.$$

Hence $\mathrm{Var}(Y) \leqslant (1/2 + 2\varepsilon^2 + 2\varepsilon)\sigma^2$ and (3.2) follows. □

Consider now the random variable $P(S,\varepsilon)$ when S is chosen randomly. Define a sequence of random variables $X_0, X_1, \ldots, X_n$ as follows. $\mathrm{Prob}(X_0 = 0) = 1/2$, $\mathrm{Prob}(X_0 = 1) = 1/2$. For $i \geqslant 0$, X_{i+1} is obtained from X_i as follows: let Z_1, Z_2 be two independent random variables having the probability distribution of X_i and put $X_{i+1} = (1/2+\varepsilon)\min(Z_1,Z_2) + (1/2-\varepsilon)\max(Z_1,Z_2)$. Clearly X_n is the random variable $P(S,\varepsilon)$. Since $E(X_0) = \sigma(X_0) = 1/2$ repeated application of Lemma 3.2 implies

$$E(X_n) \geqslant E(X_0) - \sqrt{2}\varepsilon \frac{1}{1-\sqrt{1/2+2\varepsilon^2+2\varepsilon}}\sigma(X_0)$$

$$= 1/2\{1 - \sqrt{2}\varepsilon/[1-(1/2+2\varepsilon^2+2\varepsilon)^2]\}$$

$$\sigma(X_n) \leqslant (1/2+2\varepsilon^2+2\varepsilon)^{n/2}\sigma(X_0) = 1/2(1/2+2\varepsilon^2+2\varepsilon)^{n/2}.$$

This proves Theorem 3.1. □

It is worth noting that we can slightly improve our bounds to show that $E > 0$ provided $1/2 + 2\varepsilon^2 + 2\varepsilon < 1$. We omit the details.

4. CONCLUDING REMARKS

We have shown that under reasonable assumptions the output bits of a single weakly random source are reasonably random. Thus, e.g., by the observation of Section 2, a $1/2 \pm (1/2\sqrt{900}) = 1/2 \pm 160$ biased coin is reasonably good, even under adversary assumptions, for checking primality of a 900-bit number using the randomized algorithm of [Ra]. Under the (plausible) assumption that the set of

witnesses is random, even a much worse coin is sufficient, by the results of Section 3.

It would be interesting to decide if $E_{n,\varepsilon}$ defined in Section 3 is bounded away from 0 for every fixed $\varepsilon > 0$.

ACKNOWLEDGMENTS

We would like to thank Johan Hastad for fruitful discussions. Research supported in part by Allon Fellowship and by the Fund for Basic Research administered by the Israel Academy of Sciences and in part by NSF Grant CCR-8704513 at Harvard University.

REFERENCES

[Bl] M. Blum, "Independent unbiased coin clips from a correlated biased source: A finite state Markov chain," *Proc. 25th FOCS, Florida* 425–433 (1984).

[CG] B. Chor and O. Goldreich, "Unbiased bits from weak sources of randomness," *Proc. 26th FOCS, Portland, Oregon* 429–442 (1985).

[El] P. Elias, "The efficient construction of an unbiased random sequence," *Ann. Math. Statist.* 43: 865–870 (1972).

[F] W. Feller, *An Introduction to Probability Theory and Its Applications*, 2nd ed., Vol. 1. John Wiley, New York, 1965.

[Mu] H. F. Murray, "A general approach for generating natural random variables," *IEEE Trans. Comput.* C-19: 1210–1213 (1970).

[Ra] M. Rabin, "Probabilistic algorithms," In *Algorithms and Complexity* (J. Traub, ed.), pp. 21–39 Academic Press, New York, 1976.

[SV] M. Santha and U. V. Vazirani, "Generating quasi-random sequences from slightly random sources," *Proc. 25th FOCS, Florida* 434–440 (1984).

[TRV] D. Tyger, M. Rabin, and V. V. Vazirani, in preparation.

[V] U. V. Vazirani, "Towards a strong communication complexity theory or generating quasi-random sequences from two communication slightly random sources," *Proc. 17th STOC, Providence, RI* 366–378 (1985).

[vN] J. von Neumann, "Various techniques used in connection with random digits," Notes by G. E. Forsythe, National Bureau of Standards, *Appl. Math. Ser.* 12: 36–38 (1951). Reprinted in von Neumann's collected works, Pergamon Press, New York, 1963, pp. 768–770.

[VV] U. V. Vazirani and V. V. Vazirani, "Random polynomial time is equal to slightly random polynomial time," *Proc. 26th FOCS, Portland, Oregon* 417–428 (1985).