KOLMOGOROV INFORMATION AND VLSI LOWER BOUNDS

Robert R. Cuykendall

December 1984
CSD-840052

UNIVERSITY OF CALIFORNIA
Los Angeles

Kolmogorov Information and VLSI Lower Bounds

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

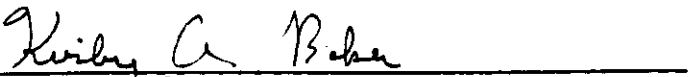
in Engineering

by

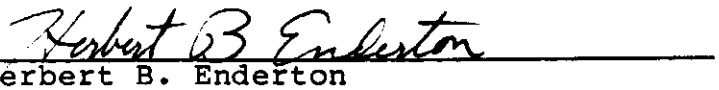
Robert R. Cuykendall

1984

The dissertation of Robert R. Cuykendall is approved.



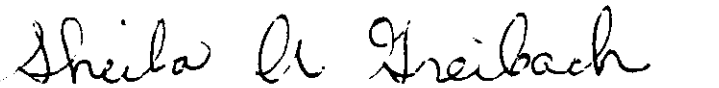
Kirby A. Baker



Herbert B. Enderton

D. Stott Parker

Jack W. Carlyle, Committee Co-Chair



Sheila A. Greibach, Committee Co-Chair

University of California, Los Angeles

1984

TABLE OF CONTENTS

LIST OF FIGURES	iv
ACKNOWLEDGEMENTS	v
VITA/PUBLICATIONS	vi
ABSTRACT	viii
1. INTRODUCTION	1
1.1 Background	1
1.2 Approach and New Results	4
1.3 Order Notation	8
2. APPROACH	10
2.1 Kolmogorov Information	10
2.2 Time Lower Bounds	13
2.3 Space-Time Lower Bounds	16
2.4 VLSI Lower Bounds	17
Background and Notation	17
New Results	26
2.5 Examples	40
3. APPLICATIONS	52
3.1 Self-Modifying Programs	52
3.2 Probabilistic Algorithms	53
3.3 Approximation Circuits	58
4. FEASIBLE VLSI COMPUTATIONS AND SPEEDUP	65
4.1 Information Theoretic Parallel Computation Thesis	65
4.2 K_ψ Characterization of Speedup	71
4.3 1-Degrees and K_ψ	78
5. MEASURE INDEPENDENCE	85
6. SOFTWARE METRICS AND VLSI LOWER BOUNDS	89
7. CONCLUSION	95
7.1 Open Problems	95
7.2 New Directions	97
REFERENCES	102

LIST OF FIGURES

Figure 1. elimination of a crossover by use of "exclusive or" gates 25

Figure 2. deterministic computation of f 49

Figure 3. γK_{ψ}-state nondeterministic computation of f 50

Figure 4. behavior of R(d,p,e) with d = 0 and e fixed 62

Figure 5. behavior of R(d,p,e) with d = 0 and p fixed 62

Figure 6. binary entropy function H 62

Figure 7. comparison of 1-d and 1-H(e) 62

Figure 8. results of a distant-correlation experiment testing the foundations of microphysics 100

ACKNOWLEDGEMENTS

The author would like to express his appreciation to Professor H. T. Kung of Carnegie Mellon University for encouraging discussions and important remarks on the approach and results of this research. Special thanks are due to Professor Sheila A. Greibach for numerous suggestions that helped in clarifying and reorganizing the dissertation, and to Professors Jack W. Carlyle and Herbert B. Enderton for many helpful discussions. The author also wishes to express his gratitude to the Jet Propulsion Laboratory of the California Institute of Technology for research support in part under NASA contract NAS7-100.

VITA

September 25, 1937	Born, New York City
1958	BSc., Massachusetts Institute of Technology
1958-1960	Teaching Assistant, Department of Mathematics, Cornell University
1960-1964	U.S. Naval Officer, Seventh Fleet
1964-1978	Technical Staff (Hughes Aircraft Co., TRW Inc. and Honeywell Inc.)
1974	M.S., University of California, Los Angeles
1978-	Staff Scientist, Jet Propulsion Laboratory, California Institute of Technology

PUBLICATIONS

R. Cuykendall, **Elementary Proofs of the Sylow Theorems, Direct-Product Theorems for Abelian and Nilpotent Groups, and Enumeration Theorems of p-Groups**, Bachelor's Thesis, Massachusetts Institute of Technology, May 26, 1958

R. Cuykendall and A. Konheim, **Coding for Non-Synchronous Channels**, G.E. Technical Information Series (Advanced Electronics Center at Cornell University), No. R59ELC95, Nov. 1959

R. Cuykendall, **Kolmogorov Complexity and Performance Measurement,** Computer Science Department Research Seminar Notes, University of California, Los Angeles, 1973

_____, **Predicting Debugging Time,** Honeywell Software Productivity Symposium, April 1977, Minneapolis, MN

_____, **Models for Limiting Debugging Resources to Simulation Coding Goals,** Proceedings Tenth NTEC/Industry Conference, Nov. 1977, Orlando, FL

_____, **Development and Management of Software Product Assurance,** Honeywell Software Management Techniques Conference, March 28-30, 1978, Minneapolis, MN

_____, **Formal Estimation of Partial Debugging Times,** Third International Conference on Software Engineering, Abstract, May 1978, Atlanta, GA

_____, **Relations Among Software Interconnectivity Metrics, VLSI Lower Bounds and Programming Primitives,** Computing Memorandum No. 490, Jet Propulsion Laboratory, California Institute of Technology (Sept. 13, 1982)

_____, **On-Board Computing Technology,** IEEE National Telesystems Conference, Nov. 1982, Galveston, TX

_____, **Software Metrics and Lower Bounds,** VLSI and Software Engineering Workshop, IEEE Computer Society Press, 1983

R. Cuykendall, et al, **Design Synthesis and Measurement,** VLSI and Software Engineering Workshop, IEEE Computer Society Press, 1983

_____, **Design Synthesis in VLSI and Software Engineering,** J. Systems and Software, Vol. 4, No. 1, 1984

R. Cuykendall, **Quantum Mechanical Models and Computational Complexity,** Technical Report D-1641, Jet Propulsion Laboratory, California Institute of Technology (June 7, 1984)

ABSTRACT OF THE DISSERTATION

Kolmogorov Information and VLSI Lower Bounds

by

Robert R. Cuykendall

Doctor of Philosophy in Engineering

University of California, Los Angeles, 1984

Professor Sheila A. Greibach, Co-Chair

Professor Jack W. Carlyle, Co-Chair

By using a source encoding result, lower bounds for the VLSI complexity of approximately computing certain uniform sets of finite functions f are related directly to the Kolmogorov information measure $K_{\psi}(f)$. This leads to a hierarchy of infinite sequences of finite functions that are increasingly amodular, which cannot be reasonably approximated by (any VLSI-efficient) modular functions. A result on the size of programs admitting speedups provides convincing evidence that practically all computations are amodular. Examples are constructed and limitations shown to the computing power of self-modifying programs, and to the reduction in VLSI complexity afforded by probabilistic algorithms or approximation circuits. The results suggest a modification to the extended parallel computation thesis in terms of the information measure K_{ψ}. Using the analogy between multiple-input arrival schedules

in VLSI and off-line Turing machines, an intuitively appealing information-theoretic argument is presented that on-line Turing machine reversal speedup is possible if and only if reversal growth rate is strictly greater than order $\log_2 n+c$ on input of length n for some fixed constant c. This leads to a characterization of speedable and nonspeedable sets in terms of K_ψ relativized to $\emptyset^{(1)}$, and to a connection between algorithmic information K_ψ and information-content as measured by various index sets. An extension of this result with K_ψ relativized to $\emptyset^{(n)}$ shows there is a structural connection between the 1-degrees and algorithmic information. It is also shown that VLSI lower bounds provide absolute lower bounds for a class of software interconnectivity metrics. Some open problems and their impact are identified, and a direction is suggested, based on recent experimental tests of the Bell inequality, for future work in defining a physical computation model which may be strictly more powerful than VLSI at quantum limits.

1. INTRODUCTION

1.1 Background

Historically, the accepted position on what is effectively computable has been to adopt Church's thesis, viz. effectively computable equals recursive (or Turing computable, or any of the many other equivalent notions). More recently with the advent of computational complexity theory, it has been suggested that many recursive functions are far too difficult to compute to be considered effectively computable in any feasible way. The generally accepted view here has been to place a polynomial (or at worst exponential) upper bound on the computation times of recursive functions. The remaining recursive functions are then considered intractable - even for inputs of small size their computation would probably take too long using the fastest computer imaginable (if subject to the causality limitations imposed by the speed of light).

During the last half dozen years, results about what can and cannot be practically computed have had a major influence on computer science. Research on feasible computations has progressed very rapidly, and has revealed deep structural relations, achieved by efficient reductions, between different classes of computations [Hartmanis 1978]. One of the primary aims of the science of

computing is to recognize and quantify exchange relations among parameters of computation which reflect the real cost of computing. Such computational tradeoffs state a relationship between the inherent complexity of a function to be computed, and the cost of performing the computations. Research aimed at understanding the inherent complexity of computational problems has generally been based on measures which reflect such computational costs as the number of arithmetic or logical operations (time of computation), switching elements (circuit size) or storage registers (amount of memory) required.

Examination of many common computations shows that even with current algorithms there is an enormous amount of parallelism available to be exploited, and that the inherently sequential steps (built-in data dependencies for arithmetic or logical operations) can often be made small in number. Computations in which the inputs and outputs are partitioned among a number of processors must in general transfer information between the processors. Recent work in distributed computations has shown this data movement to be far more limiting to speed than data dependency [Abelson 1980, Gentleman 1978]. In some computations the processors spend significantly more time waiting for information to be transferred than in performing actual computation. Even in the relatively straightforward situation of memoryless processors arranged in highly structured

architectures, data movement rather than arithmetic operations is often the limiting factor in determining the performance of parallel computations. For more general distributed processing architectures involving arbitrary network interconnections, the situation is correspondingly more transmission dependent.

Information transfer can be regarded as a measure of inherent modularity: a computation is inherently amodular if any way of partitioning the computation demands highly interacting parts. This amodularity entails an area-time tradeoff for VLSI circuits. In VLSI chips the computation is distributed over the chip, and the various processing elements must communicate via wires whose area is usually as large (up to a constant) as the area of the layout [Yao 1981]. Such is the case in computing the discrete Fourier transform, integer multiplication, permutation, matrix multiplication, sorting and in fact for any computation involving the notion of shifting or transitivity [Vuillemin 1980]. This state of affairs is independent of circuit interconnection pattern and holds for all VLSI layouts, including non-deterministic layouts if they could be built [Lipton-Sedgewick 1981]. Thus at one extreme VLSI technology will allow conventional architectures to be implemented as smaller, faster and cheaper machines through more sophisticated interconnections of machine components. At the other extreme, hierarchic VLSI architectures have been proposed that are

radical departures from the von Neumann (sequential) architecture [Backus 1978, Mago 1979, Wilner 1978]. While dramatic improvements in the performance of integrated structures can be achieved by such hierarchical organization, a penalty is always paid in the area required for wires.

1.2 Approach and New Results

The information-theoretic approach taken in the present work is described in section 2. We begin with definitions in section 2.1 introducing the concept of the intrinsic descriptive complexity K_ψ of a finite object based on the early work of Kolmogorov and Solomonoff, and briefly review in section 2.2 its recent use in obtaining lower bounds on dynamic complexity measures such as running time. An earlier application of descriptive complexity to the problem of formulating inequalities involving the computational parameters space and time in serial computing structures is sketched in section 2.3.

We begin section 2.4 with a review of concepts and techniques for deriving lower bounds for VLSI computations. Using a source encoding result due to T. Fine we show in Theorem 1 that there exists an infinite sequence of sets M_n of binary functions such that each M_n contains at least one function f not in C, where C is the

set of reasonably programmable binary functions in the sense that there exists a program P which yields an approximation $f' \varepsilon \delta(f)$ to f, that P run in time no longer than τ, and that $|P| < \gamma[K_\psi(f)]$ for all pre-assigned recursive τ, γ and δ, γ-increasing and δ such that for some $\varepsilon < 1$ and each binary function $g \varepsilon M_n$, $\delta(g)$ contains no more than $2^{\varepsilon|g|}$ binary functions.

Next we relate lower bounds on the VLSI complexity of approximately computing certain uniform sets of finite binary functions f directly to their information content $K_\psi(f)$. We show in Theorem 2 an infinite sequence of sets M_n of binary functions such that each M_n contains at least one function f which cannot be computed or approximated within $\delta(f)$ by any VLSI circuit in less than AT^2 (equivalently A^2T) $> \Omega[h(f)]$ or AT^2 (equivalently A^2T) $> \Omega[h(K_\psi)]$, for all recursive h, and recursive δ such that for some $\varepsilon < 1$ and each binary function $g \varepsilon M_n$, $\delta(g)$ contains no more than $2^{\varepsilon|g|}$ binary functions.

An interpretation of Theorem 2 suggests a hierarchy theorem (Theorem 3) in which we show the existence of infinite sequences of finite functions f of increasing degree of amodularity which cannot be reasonably approximated to within $\delta(f)$ by (VLSI-efficient) modular functions. We show in Corollary 1 that if all sufficiently large M_n contain at least one function f not in C, then every

(reasonably accurate) computation allowing arbitrary inputs over $\Sigma^{n \geq n_0}$ is inherently amodular. Using a result due to J. Helm and P. Young on the size of programs admitting speedups, Theorem 4 (in which we show that Theorem 1 extends to all sufficently large M_n if the Helm-Young result extends to all sufficiently large operators R) provides convincing evidence that practically all computations are amodular.

In section 2.5 naturally occurring examples are constructed, and several complexity-theoretic properties are identified which can evidently lead to the difficult-to-compute sequences in Theorems 1 - 3, e.g., the property of being retraceable, speedable, having an arbitrarily low accessible information rate, or computation consisting of an infinite set of easily proved theorems that takes arbitrarily long to recognize.

We apply these theorems in section 3 to show limitations to the computing power of self-modifying programs, and to the reduction in VLSI complexity afforded by stochastic algorithms of either Rabin or Karp type, or by approximation circuits as defined by Pippenger. In section 4.1 we use Theorem 1 to show limitations in terms of K_ψ on the efficient simulation of Turing machine space and reversal. In Theorems 5 and 6 we show there are infinitely many finite functions f with polynomially bounded (respectively τ-bounded)

space and reversal such that parallel computation takes unbounded hardware or time in terms of information content $K_\psi(f)$ if f runs in polynomial time (respectively $\tau(|P|)$ time, τ any recursive function) or takes longer than any recursive function of input length $|P|$. These theorems suggest a modification to the extended parallel computation thesis of Cook and Dymond in terms of the information measure K_ψ.

Using the analogy between multiple-input arrival schedules in VLSI and off-line Turing machine computation, an intuitively appealing information-theoretic argument is presented in Theorem 7 that on-line Turing machine reversal speedup is possible if and only if reversal growth rate is strictly greater than order $\log_2 n+c$ on input of length n for some fixed constant c independent of n. This provides an intuition behind the extended parallel computation thesis which leads in section 4.2 to a characterization of speedable and nonspeedable sets in terms of K_ψ relativized to $\emptyset^{(1)}$ (Theorem 8), and to a connection in section 4.3 between the algorithmic measure K_ψ and information content with respect to the 1-degrees as measured by the index set FIN (Theorem 9). An extension of Theorem 9, with K_ψ relativized to $\emptyset^{(n)}$, establishes a general structural connection between the 1-degrees and algorithmic information (Theorem 10).

Section 5 provides evidence that the approach herein with respect to

a number of different information measures, including the Shannon entropy, is measure-independent. It is shown in Theorem 11 (section 6) that VLSI lower bounds are in fact lower bounds for a class of software interconnectivity metrics which arise in a natural way from the VLSI implementation of programs, and hence provide a method for assessing the validity of software complexity metrics. Thus differences arising between VLSI and software measures should asymptotically disappear as programs for deriving circuits become more common. Earlier versions of this material have appeared in [Cuykendall 1982a, 1982b, 1982c, 1984a].

In section 7 we discuss some open problems and propose a direction for future work in defining a physical computation model which may avoid some of the limitations shown in our Theorems 1-6. Our proposal, inspired by recent experimental tests of the Bell inequality, suggests that transmissions which violate separability might sustain a computation and thus establish a physically defined complexity class strictly more powerful than VLSI at quantum limits.

1.3 Order Notation

We use the following asymptotic function order notation:

Let f, g be functions defined on the set of natural numbers.

$f(n) = O(g(n))$ $f(n)$ is order at most $g(n)$ if and only if there are constants $c > 0$ and $n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

$f(n) = \Omega(g(n))$ $f(n)$ is order at least $g(n)$ if and only if there are constants $c > 0$ and $n_0 > 0$ such that $f(n) \geq cg(n)$ for all $n \geq n_0$.

$f(n) = \theta(g(n))$ $f(n)$ is order exactly $g(n)$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

2. APPROACH

2.1 Kolmogorov Information

As a consequence of the work of [Kolmogorov 1965] and [Solomonoff 1964], a notion of the intrinsic (static) descriptive complexity of a finite object has been developed. Intuitively, descriptional complexity is defined as the minimum number of binary signs containing sufficient information about a given object for its recovery (decoding). This definition depends essentially on the method of decoding. Thus, some computers are easier to program than others, or equivalently some programming languages (description schemes) are more efficient than others. However, the existence of a universal Turing-Post machine, or optimal programming language (which simulates running a program on another computer when it is given a description of that computer together with its program) is known from recursion theory (or computability theory), and an invariant definition of the descriptive complexity of an object can therefore be given.

We consider two related measures of information content introduced by [Kolmogorov 1965]. Let s denote a finite binary string and let x denote an infinite binary sequence. The first n bits of x are written as x^n, x_n (sometimes written x(n)) is the n^{th} bit of x

and $\ell(s)$ is the length of s. We also write x^n to denote an arbitrary finite string of length n.

Definition. A programming language φ is a partial recursive function on the finite strings,

$$\varphi: \{0,1\}^* \longrightarrow \{0,1\}^*.$$

Definition. p is a φ-program for s iff $\varphi(p) = s$.

Intuitively, programming languages (computers) are thought of as mappings from programs to their outputs. The value $\varphi(p)$ is the binary string output by the computer φ when it is given the program p. If $\varphi(p)$ is undefined, this means that running the program p on φ produces an unending computation with no output.

Definition. The Kolmogorov information in x^n relative to the programming language φ is

$$K_\varphi(x^n) = \min \{\ell(p): \exists p\ \varphi(p) = x^n\}$$

$$= \infty \text{ otherwise.}$$

Definition. ψ is an optimal programming language iff

$$\forall \varphi \; \exists c \; \forall s \; [K_{\psi}(s) \leq K_{\varphi}(s) + c].$$

The existence of optimal programming languages is well known [Rogers 1967]. Such languages ψ have the property that for any programming language φ, there is a constant c (which depends on φ) such that the shortest programs relative to ψ never exceed the shortest programs relative to φ by more than c, independent of the string s being programmed. Thus, ψ is as succinct a relative description scheme as any. Therefore, we define the Kolmogorov information measure simply as K_{ψ}. Kolmogorov information content is often equivalently referred to as algorithmic information content, program-size complexity, Kolmogorov complexity or descriptive complexity, and inputs (programs) p for which $K_{\psi}(p) \geq |p|$ are called incompressible.

It has also been found useful to study programs which are given, as a separate input, the length of the desired output, where no charge is made for the length of this second input.

Definition. The Kolmogorov conditional information in x^n is

$$K_{\psi}(x^n|n) = \min \{\ell(p): \exists p \; \psi(p,n) = x^n\}$$

$$= \infty \text{ otherwise.}$$

The two measures express a slightly different quality of the sequence x^n in assessing its information content. The quantity $K_\psi(x^n)$ gives the minimum length of programs for x^n which must contain, in addition to the distribution of 1's and 0's in x^n, also information concerning the length n. The integer n can generally be expected to use about length $\log_2 n$ of the binary program p for x^n. On the other hand, the quantity $K_\psi(x^n|n)$ gives the minimum length of a program which need not contain information on the length n, but which only determines the distribution of 1's and 0's in x^n. This distinction is dramatic at the low-information end of the scale where the information needed to determine the distribution is less than $\log_2 n$.

2.2 Time Lower Bounds

Recent developments in the study of the structure of feasible computations have established close links between what can and cannot be proven about running times of algorithms, and the problem of establishing sharp time bounds for one-tape Turing machine computations (separation of complexity classes) [Hartmanis 1977, DeMillo-Lipton 1979]. Thus, the inability to prove running times for algorithms is related to the existence of gaps in the hierarchy of complexity classes. (It is shown in [Hartmanis 1977] that the Gap Theorem for resource bounded computations [Borodin 1972] can hold only for non-constructively defined computational complexity

classes.) [Chaitin 1976b] has shown complexity gaps exist in both the Kolmogorov-Solomonoff descriptional complexity and the process complexity measure of Schnorr. Thus, one expects similar difficulties with such measures.

Nevertheless, the static descriptional complexity of inputs to algorithms can be used to obtain lower bounds on dynamic computational complexity, such as running time. While most algorithms treat numbers (eg. to be sorted) as something atomic, in the sense that whole numbers can be compared or transported, it is not allowed to break the binary representation of numbers into parts and to perform computations with the fragments of the numbers. Analysis in terms of the Kolmogorov information measure at least partially removes this restriction.

Lower bounds on inherent worst-case computational complexity are typically easy to conjecture but hard to prove. One reason is the difficulty in finding sufficiently hard inputs for each different algorithm. The worst case for one algorithm might be expedited as a special case by another. Thus, if we can find inputs not susceptible to handling as special cases, then we might be able to convert out intuitions to proofs more easily.

One way to handle an input efficiently as a special case is to find a much smaller description of the same input. We can prevent this

sort of special handling if inputs are provided which are already suitably incompressible. The work of Kolmogorov and Solomonoff shows how to make this precise, and that suitably incompressible streams of input data are abundant. The incompressibility forces simulators to use a lot of space, and hence to spend a lot of time retrieving distant information. The information-theoretic approach also serves as a rigorous, yet natural, tool equivalent to vague intuitions already in limited use. In this sense, its value is analogous to that of non-standard analysis. These potentially valuable intuitions have not been cultivated much in the past, because conversion to rigorous proofs seemed so difficult. The new approach is to look at a particular sequence which is random in the rigorous, domain-independent sense that it is incompressible. The effect is to remove complicated and obscuring domain dependent counting from such proofs, and instead simply to cite the result of the one simple counting argument which shows that there are incompressible strings.

[Paul 1979] used the information-theoretic approach to obtain restricted lower bounds on the time complexity of sorting. [Paul, Seiferas and Simon 1980] used the technique to obtain nonlinear lower bounds on the time complexity of simulating abstract storage units on-line. Improved lower bounds on sorting and simplified error estimations for probabilistic algorithms have been attained in

[Reisch-Schnitger 1982] by analysis in terms of Kolmogorov information.

2.3 Space-Time Lower Bounds

[Savage 1973] and [Cuykendall 1973] independently proposed that a measure of the descriptional complexity of functions be applied to the problem of formulating inequalities involving computational parameters, leading to lower bounds on the tradeoff between storage and time on serial computing structures. The procedure employed to derive computational inequalities is as follows. Assume that a function f is computed by a serial computer with S bits of storage in T cycles. Then the T cycles can be simulated on a 1-tape universal Turing machine ψ, and the simulation program will be no shorter than the descriptional complexity of f. Once a good upper bound to the length of the simulation program has been derived, an inequality involving parameters of computation such as S, T and the K_ψ complexity of f can be established. The role of the 1-tape Turing machine in this process is that of a canonical framework to which computations on a serial computer can be translated for the purpose of deriving computational inequalities.

Combinational measures $C_b(f)$, which count the number of steps in a circuit C using straight line algorithms relative to some basis operations b (no branching or looping, equivalent to a switching

circuit) to compute f, are in general less constraining to the trade region, i.e.

$$K_{\psi}(f) \leq k_1 C_b(f) \log C_b(f) + k_2$$

for constants k_1, k_2 independent of f, as shown in [Savage 1973].

2.4 VLSI Lower Bounds

Background and Notation

In order to compute a family of functions in which the inputs and outputs are distributed among a number of processors, information must in general be transferred between the processors. The role of such internal communication requirements in contributing to the inherent complexity of computational problems is still poorly understood. In distributed systems it can be expensive both in time and hardware to send information between processors, and in some computations the processors spend significantly more time waiting for information to be transferred than in performing actual computation. In VLSI chips the computation is distributed over the chip, and the various processing elements must communicate via wires. These wires generally occupy more space than the processors

themselves, and can therefore be a more significant factor in determining cost and performance [Mead-Rem 1979].

The information transfer required in a distributed computation is defined to be the smallest number I such that, for any values of the inputs, the computation can be accomplished with a total of at most I units of information transferred between the processors. In general, the information transfer can be regarded as a measure of the inherent modularity of the function being computed. Finding a configuration for which the inputs or outputs are evenly distributed, but which requires small information transfer, is a way of modularizing the computation. If this is not possible, we say that the function is inherently amodular. In other words, any partitioning of the computational process demands highly interacting parts. This (information transfer) amodularity leads to an area-time tradeoff for VLSI circuits.

The basic model of VLSI computation allows great generality. It allows features which certainly are not even contemplated in the near future. There are three main components: the boolean function f which is to be computed, a synchronous circuit C that computes f, and a VLSI layout V that realizes C. We assume that C is a network of wires attached to each other and to gates. The gates of C can be "and", "or" or "not" gates of arbitrary fan-in and fan-out. Such a circuit, which may have feedback, computes f provided there is an

input-output schedule that describes how the inputs and outputs of f are mapped onto the input and output wires of C. It is assumed that each input arrives once and each output leaves once. (The motivation for this is that otherwise we would be allowing the circuit "free" memory.)

Definition. An input-output schedule is where-oblivious (when-oblivious) provided where (when) the inputs arrive and the outputs leave is independent of the values of the data.

(Note that of the two types of schedules the first is the more common since when inputs arrive often depends on both when and what particular addresses are generated, whereas scheduled inputs to a processor typically go to fixed locations.)

Definition [Lipton-Sedgewick 1981]. A VLSI layout V is a (λ, μ_1, μ_2)-layout of the sequential circuit C if there is a map that assigns to each gate g (wire w) of C a closed connected region of the plane g* (w*) so that

(1) if w is an input or output wire of gate g, then g* intersects w*, and

(2) for each $\lambda \times \lambda$ square S of the plane,

(a) at most μ_1 gates g map to regions g* that intersect S, and

(b) at most μ_2 wires w map to regions w* that intersect S.

It is further assumed that all of the g* and w* lie in a convex region R, the region of the layout.

No assumption is made about the location of circuit inputs or outputs or how they are assigned to gates, but we do require that circuits use all their inputs. Condition (1) simply forces electrical connections to map to topological connections (the converse is not true because multiple layers are allowed). Condition (2) is a direct result of the limits of VLSI fabrication. It ensures that any such S (e.g., a transistor) can only "see" a fixed number of gates and wires, and therefore limits the number of layers that can be used at the same point to $\mu_1 + \mu_2$.

The first technique [Thompson 1979] used to prove lower bounds for VLSI layouts involved a cut theorem relating layout area A and time T to the information flow across a line (the bisection-width) which divides the layout into two parts. The layout area A for a function f is defined as the area of the smallest region R containing a VLSI layout V of a network C of gates and interconnecting wires which

computes f in time T under a fixed scaling. Scaling refers to the minimum feature size λ of the implementing technology, taken as either the narrowest channel (wire) width or the area λ^2 of the smallest transistor. The idea is roughly as follows.

Draw a line which divides the layout into two parts, with about half of the inputs n to the circuit in each part. Some restrictions about the geometry of the layout and the nature of the inputs are needed to ensure that this can be done. If the line cuts through about w wires (the line could also cut through gates, a complication which must be handled carefully), then with some assumptions about the geometry of the layout and the line, it is possible to show that $A > \Omega(w^2)$. Furthermore, the value w can be thought of as a bound on the information that can flow across the boundary. If the total amount of information that must flow across the boundary during the entire computation is I, then the time taken must satisfy $T > I/w$. This leads immediately to the bound $AT^2 > \Omega(I^2)$. The proofs for specific problems are completed by showing that $I > \Omega(n)$ for any division of the circuit that puts half the inputs on either side of the dividing line.

This line of reasoning is similar to the crossing sequence arguments introduced by [Hennie 1965,1966] for one-tape Turing machine computations. Proving lower bounds for AT^2 is reduced to constructing appropriate sets of input assignments. The "planarity"

(layer) restriction of VLSI is roughly analogous to the one-tape restriction for Turing machines. The lower bound proofs are not identical however, because for VLSI the result must be proved for any possible division of the inputs into two equal parts, while for Turing machines a single division is inherent in the problem. Thus, while some computations may be hard for one-tape Turing machines, say requiring time at least $\Omega(n^2)$, there may be a VLSI layout for the computation with AT^2 at most $O(n)$. The reason for this difference is clear: for some partitions a great deal of information must flow while for others very little is needed.

A fundamentally different approach [Leighton 1981] uses crossing number and wire area arguments to find lower bounds on the layout area and maximum edge length of a variety of computationally useful networks. The crossing number of a graph is the minimum number of pairs of edges which must cross in any planar drawing of the graph. The wire area of a graph is the minimum amount of wire which is needed to lay out the graph in the VLSI model. Clearly the crossing number and wire area are lower bounds on the layout area of any graph. It is shown that

$$\Omega(w^2) \leq c + N \leq W \leq A$$

for any N-node graph with bisection-width w, crossing number c, wire area W and layout area A. This implies that every lower bound

technique for the bisection-width of a graph is also a lower bound technique for its crossing number and wire area. Thus nothing is lost by concentrating instead on finding good lower bounds for the crossing number and wire area of a graph. In fact, improved lower bounds are shown for certain separators. (An N-node graph has an f(N)-separator if it can be partitioned into two equal-sized subgraphs G_1 and G_2 such that at most f(N) edges link G_1 to G_2, and both G_1 and G_2 have f(N/2)-separators.)

One can also relate AT^2 to a notion of boolean circuit complexity. A boolean circuit is an acyclic directed graph such that each vertex has fan-in zero or two, the predecessors of each vertex are ordered, and corresponding to each vertex v of fan-in two is a binary boolean operation b_v. With each vertex of the circuit we associate a boolean function which the vertex computes, defined as follows. With each of the k vertices v_i of fan-in zero (inputs) we associate a variable x_i and an identity function $f_{v_i}(x_i) = x_i$. With each vertex w of fan-in two having predecessors u, v we associate the function $f_w = b_w(f_u, f_v)$. The circuit computes the set of functions associated with its vertices of fan-out zero (outputs). For example, we can have arbitrary fan-out and allow the constants {true, false} as inputs. By associating true with '1' and false with '0', we think of every boolean circuit as realizing a function $f:\{0,1\}^n \rightarrow \{0,1\}$.

The size of a circuit C is defined as the number of interior vertices or logic gates (ie. noninput nodes), and the depth of C is the length of the longest path in C. It is common to encode a circuit C as a string in $\{0,1\}^*$. This can be done in a straightforward way; e.g. topologically order the network, give addresses to each of the vertices, and then give the circuit as a sequence of instructions. Clearly every circuit of size C(f) can be encoded into a 0,1 string of length at most polynomial in C(f).

Any boolean circuit can be converted into a planar circuit by the following steps. First embed the circuit in the plane, allowing edges to cross if necessary. Next, replace each pair of crossing edges by the crossover circuit shown in Figure 1. It follows that any lower bound on the size of planar circuits which compute a certain function f is also a lower bound on the total number of vertices and edge crossings in any planar representation of a non-planar circuit which computes the function f. In fact, it is shown in [Savage 1981] that planar circuit complexity (size) is no larger than quadratic in the standard circuit complexity, and that for most boolean functions (actually binary functions) standard and planar circuit complexity measures have about the same value.

Any VLSI layout can be simulated by a boolean planar circuit, the only difficulty being that the layout may use feedback, and this is not allowed in the planar circuit. Moreover, if P(f) denotes the

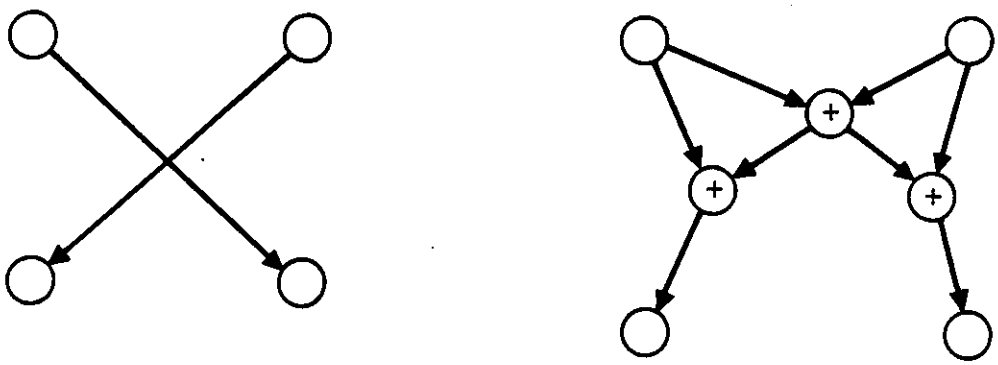


Figure 1. elimination of a crossover by use of "exclusive or" gates

size of the smallest planar boolean circuit for f, then $AT^2 > \Omega[P(f)]$ for when and where oblivious layouts of area A and time T [Lipton-Sedgewick 1981]. Thus, the P(f) results obtained in [Lipton-Tarjan 1980] by the efficient application of divide-and-conquer to problems on planar graphs can be used to get lower bounds on AT^2.

A slightly more general result appears in [Savage 1981], that for all functions $f:\{0,1\}^n \rightarrow \{0,1\}^m$ both $AT^2 > \Omega[P(f)]$ and $A^2T > \Omega[P(f)]$ for when and where oblivious layouts when the chip algorithm not only allows input variables to be read multiple times, but can read input multiple times at more than one place on the chip. It is also noted that there is a connection between the second inequality and lower bounds to space and time for

uniprocessor machines. Namely, the separability analysis used by [Grigoriev 1976] can be extended to the VLSI model to obtain lower bounds to A^2T, and thus all previous bounds obtained with that method [e.g., Savage-Swamy 1979] apply here. We therefore have a variety of techniques for obtaining lower bounds on combinational circuit measures which can be used to establish lower bounds on the VLSI measures A^2T and AT^2. However, in general these results are weaker than those obtained by crossing sequence techniques, and therefore also weaker than those obtained by either wire-area or crossing-number arguments.

New Results

An extension of the ideas used to derive lower bounds from minimal input programs (sections 2.2 and 2.3) can be applied to determine further constraints on the area-time tradeoff relation for VLSI circuits. The concept of partitioning a computation among several processors is extended to partitioning a reasonably incompressible input program, representing several partial computations, among the processors in such a way that the information transfer required between processors (effectively, running time) is related directly to the program size. This leads to a new technique for establishing lower bounds on area-time tradeoffs for VLSI.

The concern in information theory with only average program (codeword) length has obscured the problem of the efficient coding (optimal programming) of individual message sequences. A theorem due to T. Fine demonstrates that any code that can be realistically decoded must, for many sources, assign incredibly long (exceeding $\gamma[K_\psi]$) programs P to some (not very many) messages relative to the minimum possible (incompressible) codeword length K_ψ. This limitation seems to be fundamentally unavoidable even under very weak assumptions concerning the definition of a running time bound, the concept of a neighborhood set of a sequence, and the degree γ to which the length of a program approaches its minimal value K_ψ. The concern with individual messages is formulated and treated within the framework of algorithmic information theory, rather than with respect to sources for which a relative frequency characterization of uncertainty is assumed.

The ability to compress binary data in the form of a binary sequence, for example, requires a compression (encoding) function E and a reconstruction (decoding) function ψ. If $|S|$ denotes the length of the data sequence S, it is hoped that $|E(S)|$ tends to be less than $|S|$ at least for those sequences in a finite message source M. The fundamental difficulty of data compression dealt with concerns ψ rather than E, the properties of the decoder rather than those of the encoder. The nature of the conflict is such that for many message sets (equivalently, classes of boolean functions) it is

practically impossible to decode some efficiently encoded sequences (incompressible or nearly incompressible input programs) from either probabilistic, non-probabilistic or unknown sources. There is a similar conflict between the degree to which a sequence is compressed, and the difficulty of doing so. Details of this argument are analogous to the first problem. Furthermore, the results remain valid even when the coding requirements are relaxed so that the decoder need reconstruct only a reasonable approximation to the encoded sequence.

Specifically, if $\tau(S)$ is the running-time bound on computational effort of decoder (receiver-computer) ψ accepting codeword (program) P for message S, and $\gamma[K_{\psi}(S)]$ is the upper bound to acceptable codeword length $|P|$ when the shortest codeword for S has length $K_{\psi}(S)$, then for many message sources M there exist messages $S \varepsilon M$ such that

1) if encoder satisfies γ, then decoder violates τ

2) if decoder satisfies τ, then encoder violates γ.

These conclusions seem to be fundamentally unavoidable, and remain valid even when the decoder is allowed to reconstruct only an approximation S' in a neighborhood $\delta(S)$ of S. Compatibility of these results with those of information theory is that detailed

properties of coding systems for individual messages, and not ensemble average properties, are analyzed. In a sense, concepts of Kolmogorov [Chaitin 1974, 1975] increase resolving power of information theory for looking at individual sequences, and thus reveal obstacles to uniformly good coding systems.

For the case where the receiver (computer) has no description of the message source although the transmitter (encoding program) may have a complete description, which is the case of interest in the present application, it is shown in [Fine 1975] that:

Theorem. If τ, γ and δ are recursive functions, γ increasing and δ such that for some $\varepsilon < 1$ each sequence S has a neighborhood $\delta(S)$ containing no more than $2^{\varepsilon|S|}$ sequences, then

$$(\exists \ell_0)(\forall \ell > \ell_0)(\exists \xi(\ell) \varepsilon M_{|\gamma(\ell)/1-\varepsilon)|}) \xi(\ell) \varepsilon \vec{C}$$

where $M_n = \{S: |S| = n\}$

$|x|$ is the smallest integer $\geq x$

$C = \{S:(\exists P)\psi(P)\varepsilon\delta(S), \rho_\psi(P) < \tau(S), |P| < \gamma[K_\psi(S)]\}$ is the set of reasonably compressible source sequences in the sense that P yield a reasonable approximation $S'\varepsilon\delta(S)$ to S, that P run in time no longer than τ, and that it is moderately efficient in that for some pre-assigned function γ, $|P| < \gamma[K_\psi]$, where K_ψ is the

length of the most efficient possible codeword

ψ is a p.r. function which can be thought of as any one of countably infinitely many universal Turing machines using the encoding function E

ρ_ψ is the running time of P on ψ and is defined iff ψ halts

$K_\psi(S) = \min \{|P| : \psi(P)=S\}$, the descriptive complexity of S

The above result thus shows that there is an infinite sequence $\{M_{|\gamma(k)/1-\varepsilon|}\}$ such that each $M_{|\gamma(k)/1-\varepsilon|}$ contains at least one sequence $\xi(k)$ not in C. This conclusion can be extended to other sequences of subsets of $\{0,1\}^*$, but it is not known whether it extends to all sufficiently large M_n. (Based on the program-size vs speedup result announced for certain functions f in [Helm-Young 1971], we conjecture it does extend to all sufficiently large message sources M_n, but the specification of n would be noneffective.)

There is a close connection between the theory of information transmission over a channel and complexity of computing. The above

result of Fine can be extended to binary functions which leads to the following theorem.

Definition. $M_n = \{f:|f|=n\}$ and $C = \{f:(\exists P)\psi(P)\varepsilon\delta(f),\ \rho_\psi(f) < \tau(f),\ |P| < \tau[K_\psi(f)]\}$ is the set of reasonably programmable binary functions in the sense that there exists a program P which yields an approximation $f'\varepsilon\delta(f)$ to f, that P run in time no longer than τ, and that $|P| < \gamma[K_\psi(f)]$ for all pre-assigned recursive τ, γ and δ, γ-increasing and δ such that for some $\varepsilon < 1$ and each binary function $g\varepsilon M_n$, $\delta(g)$ contains no more than $2^{\varepsilon|g|}$ binary functions.

Theorem 1. There is an infinite sequence of sets M_n of binary functions such that each M_n contains at least one function f not in C.

Proof. The table of an arbitrary binary function $f:\Sigma^{\log n}\rightarrow\Sigma^m$, where $\Sigma = \{0,1\}$, can be regarded as a binary sequence $S\varepsilon M_{mn}$ for some message source M_{mn} by concatenating the n rows of length m comprising the table. The encoding E(S) is then a program P for computing f on (universal Turing machine) ψ. By the previous theorem there is an infinite sequence $\{M_{|\gamma(k)/1-\varepsilon|}\}$ such that each $M_{|\gamma(k)/1-\varepsilon|}$ contains at least one binary function $\xi(k) = f$ not in C.

result of Fine can be extended to binary functions which leads to the following theorem.

Definition. $M_n = \{f:|f|=n\}$ and $C = \{f:(\exists P)\psi(P)\varepsilon\delta(f),\ \rho_\psi(f) < \tau(f),\ |P| < \tau[K_\psi(f)]\}$ is the set of reasonably programmable binary functions in the sense that there exists a program P which yields an approximation $f'\varepsilon\delta(f)$ to f, that P run in time no longer than τ, and that $|P| < \gamma[K_\psi(f)]$ for all pre-assigned recursive τ, γ and δ, γ-increasing and δ such that for some $\varepsilon < 1$ and each binary function $g\varepsilon M_n$, $\delta(g)$ contains no more than $2^{\varepsilon|g|}$ binary functions.

Theorem 1. There is an infinite sequence of sets M_n of binary functions such that each M_n contains at least one function f not in C.

Proof. The table of an arbitrary binary function $f:\Sigma^n \rightarrow \Sigma^m$, where $\Sigma = \{0,1\}$, can be regarded as a binary sequence $S\varepsilon M_{mn}$ for some message source M_{mn} by concatenating the n rows of length m comprising the table. The encoding E(S) is then a program P for computing f on (universal Turing machine) ψ. By the previous theorem there is an infinite sequence $\{M_{|\gamma(k)/1-\varepsilon|}\}$ such that each $M_{|\gamma(k)/1-\varepsilon|}$ contains at least one binary function $\xi(k) = f$ not in C.

[The restriction to binary functions in the theorem is clearly unnecessary, as any finite function of natural numbers into natural numbers can be transformed into an equivalent function f over Σ^*.]

From the known connections between Turing machine time and circuit size, and between Turing machine space and circuit depth [Borodin 1977], together with the previous observations relating circuit complexity and VLSI measures A^2T (AT^2) to P(f), we obtain the next result.

Theorem 2. There is an infinite sequence of sets M_n of binary functions such that each M_n contains at least one function f which cannot be computed or approximated within $\delta(f)$ by any VLSI circuit in less than AT^2 (equivalently A^2T) $> \Omega[h(f)]$ or AT^2 (equivalently A^2T) $> \Omega[h(K_\psi)]$, for all recursive h, and recursive δ such that for some $\varepsilon < 1$ and each binary function $g \varepsilon M_n$, $\delta(g)$ contains no more than $2^{\varepsilon|g|}$ binary functions.

Proof. By Theorem 1 we know there exists at least one f not in C for each M_n. Thus either f violates the allowed running time τ on ψ, or it violates the allowed program-size bound $\gamma[K_\psi]$ for f on ψ. If τ is exceeded for any recursive τ, then since a $\tau(n)$ time bounded Turing machine simulated on n bits by a boolean circuit requires $\theta[\tau(n)\log\tau(n)]$ gates [Pippenger-Fisher 1979, Schnorr 1976], and any planar circuit requires at least this number of gates, the

result follows with $h = [\tau \log \tau] = P(f)$. If on the other hand γ is exceeded so that any program for f takes more than $\gamma[K_\psi(f)]$ tape squares to read into storage (for example, on an off-line machine with a two-way read only input tape), more than $\gamma[K_\psi(f)]$ time steps on ψ are required just to read the input tape. This leads to precisely the same argument as before, and thus the result is established. Note that the entire input tape must be read in the computation of f, since by Theorem 1 the necessary length $|P|$ of any program for f exceeds $\gamma[K_\psi(f)]$ if running time is to remain less than $\tau(f)$, and any lesser input in computing f will violate τ. If a model of computation is assumed whereby the input is already on a working tape, either the above argument or one relating Turing machine space polynomially to planar circuit depth [Borodin 1977] leads to the stated result. (A result requiring the layout area A of any VLSI circuit performing binary multiplication be proportional to the total number of bits input to the chip appears in [Brent-Kung 1980] and is extended in [Baudet 1981] to functions corresponding roughly to shifting or transitive operations which depend on all their inputs.)

The connection pointed out earlier between on-chip information flow and inherent modularity of functions ignores the problem of input, and instead assumes each processor already has its roughly equally divided share of inputs in memory. Given enough input, no interaction among the processors would in theory be required,

regardless of the function being computed. Account for program length must therefore be taken in defining the concept of modularity. Intuitively, we think of a computation being modular if the total time it takes can be arbitrarily reduced by partitioning among a large enough number of processors. Total time in practice obviously includes the time required to transfer program bits into the various processor memories. Thus, modularity relates to all processor interactions, not just to on-chip flows. Using this global notion of processor interaction in measuring inherent modularity of computations, we can show that finite functions abound that are highly amodular. Moreover, such functions cannot be reasonably approximated by modular functions.

Definition. If a function (computation) f requires a program inefficiency (redundancy) γ, i.e.

$$|P| = \gamma[K_{\psi}(f)],$$

for f to run within time $\tau(|P|)$ on ψ, then f has amodularity degree $h = \max(\gamma,\tau)$, the greater growth rate. If h is polynomial or less, f is modular. Otherwise f is amodular.

Theorem 3 (Hierarchy Theorem). There is an infinite sequence of finite functions f of arbitrarily increasing degree of amodularity

which cannot be approximated to within reasonable $\delta(f)$ by functions f' having an amodularity degree lower in the sequence.

Proof. Theorem 1 establishes the existence of infinite sequences of finite functions that take longer than τ time to compute on ψ, or require input longer than $\gamma(K_\psi)$. Such sequences exist for each pair (τ,γ) where τ is recursive and γ is recursive-increasing. If ψ satisfies τ, then the input violates γ. The polynomial relation between Turing machine space and circuit depth [Borodin 1977] thus requires chip interaction $> \gamma(K_\psi)$ to within a polynomial factor. The degree of amodularity therefore grows as γ if τ is small. If on the other hand the input satisfies γ, then ψ violates τ. For complexity bounds $\Omega(\log n)$ on input of length n, it is well known [Dymond-Cook 1980] that sequential space and reversal are both $\Omega(\log\tau)$, and that reversal is polynomially related to aggregate depth (parallel time on combinational circuits which can re-use their gates). Thus, either the space-circuit depth or reversal-aggregate depth relation requires chip interaction $>\log\tau$ to within a polynomial factor. For the pair (τ,γ) we then have at least one (actually many) infinite sequence of functions f having degree of amodularity greater than γ or $\log\tau$ (to within a polynomial factor). By successively choosing faster growing τ and γ we obtain infinite sequences of functions with arbitrarily increasing degrees of amodularity. As ψ is required to compute (in Theorem 1) only an approximation f' to f, subject to constraints on

the reasonableness of the approximation, we can replace each f by any f' within the δ neighborhood obtaining f' sequences instead of f sequences. Clearly f' has the same degree of amodularity as f, since the running time for any such f' approximating f exceeds τ or ψ requires more than $\gamma(K_\psi)$ bits to execute f', completing the proof.

Actually, the situation may be much worse. Theorem 3 establishes only the sparse existence of functions which are increasingly amodular, namely one such f for approximately each $\gamma(n)$ with $n \geq n_0$. Thus, the distance between $\Sigma^{\gamma(n)}$ and $\Sigma^{\gamma(n+_1)}$ can be very large, and in fact becomes arbitrarily large with ever increasing γ. The high-density existence of such functions, say one for each $\gamma(n_0)+k$, $k = {}_0, {}_1 \ldots$ would seem to have serious implications for very large scale computations, e.g. solving weather prediction equations. The next two results provide conditions for a high-density hierarchy theorem.

Corollary 1. If all sufficiently large M_n contain at least one function f not in C, then every (reasonably accurate) computation allowing arbitrary inputs over $\Sigma^{n \geq n_0}$ is inherently amodular.

Proof. If the conclusion of Theorem 1 extends to all sufficiently large M_n, then by the above proof $\exists n_0$ such that there is at least one f satisfying Theorem 3 for each integer $n \geq n_0$. A computation

which allows all inputs over Σ^n includes such an f as a partial computation if $n \geq n_0$, and is therefore itself amodular even when only approximated to within δ.

There is strong evidence that the condition of Corollary 1 holds. Note that Theorem 1 implies that in order to stay within any given recursive running time bound τ, the program P for computing some f in M_n must grow more rapidly as a function of $K_\psi(f)$ than any computable γ. And since $K_\psi(f') > \gamma(\ell_0)$ where n is the smallest integer $\geq \gamma(\ell_0)/1-\varepsilon$, $|P|$ must grow faster than any computable function of n. Now as we decrease τ, the function f begins to look like a function that has an almost everywhere speedup (section 4.2) at the expense of an ever larger program. Moreover, the faster program cannot be effectively determined from the given program, nor can we effectively compute from which point on the speedup started.

Such behavior is shown in [Helm-Young 1971] to arise for certain operator speedups $R(\tau_j)(n) < \tau_i(n)$ a.e. (ordinary speedups by recursive functions $r(n, \tau_j(n)) < \tau_i(n)$ a.e. are a restricted form of operator) and is conjectured to hold for all sufficiently large operators on the basis that it should be more difficult to bound the size of programs effecting large speedups than those bounding smaller speedups. A slightly stronger result appears in [Constable-Hartmanis 1971] and independently in [Meyer-Fischer

1972], the question of extension to all total effective operators R remaining open.

Theorem 4. If the Helm-Young Theorem [1971] extends to all sufficiently large operators R, then Theorem 1 extends to all sufficiently large M_n.

Proof. Helm-Young prove that for every recursive function r there is an effective operator R only slightly larger than r, and a total recursive function f which has R-speedup

$$R(\tau_j)(n) < \tau_i(n),$$

but for which the size of the program necessary to effect the speedup increases more rapidly than any recursive bound. If their result holds for any operator R sufficiently large that

$$R(\tau)(n) \geq r(n+1),$$

then f requires such programs for all running times $\tau \leq \tau_j$.
If the size of a program necessary to effect an R-speedup cannot be effectively bounded for some R-speedable $f \in M_{n\to\infty}$, then $\forall\gamma\exists n_0$ such that its initial segments f_n of length $n \geq n_0$ require programs P_n of size $|P_n| > \gamma[K_\psi(f_n)]$, where ψ computes f therefore f_n within time $\tau_j(n)$, and Theorem 1 holds for all $\tau \geq \tau_j$ and all sufficiently

large M_n. If this behavior arises for all sufficiently large R, then Theorem 1 holds additionally for all $\tau \leq \tau_j$ and sufficiently large M_n.

In view of the fact that most simple (non-composite) functions already have been shown to have time-efficient layouts only when wire area is nearly as large as the chip, it would not have been completely unexpected to find that running time (or VLSI circuit area) for arbitrary composite functions would exceed polynomial or simple exponential limits. For example, in [Lipton-Sedgewick 1981] it is shown that certain n-input functions which are easy to compute become difficult under union or composition if each input arrives once and each output leaves once. If inputs are allowed to arrive multiple times (i.e., at many different times during the computation), this result does not hold, but if we do not allow free boundary (off chip) memory, the on-chip storage area will correspondingly increase. However, the probable abundance of well-defined functions which exceed any recursive bound on the area-time product is unexpected.

[Blum 1967] gave an axiomatic characterization of some of the properties which should be possessed by a measure of computational complexity and established the existence of speedable functions - computable functions which fail to possess optimal programs in a particularly strong sense. Recursion theorists tend to like such

functions, and computer scientists tend to consider such functions somewhat pathological. It is shown in [Alton 1976] that such pathology is rampant: there is a great diversity of behavior among the collections of run-times of different functions which do not possess optimal programs, where such diversity is gauged by certain algebraic criteria which have computational significance. Roughly speaking, these algebraic criteria concern the ways in which various functions can be intermixed to satisfy requirements that certain functions can or cannot be computed more easily than certain other functions.

2.5 Examples

In general, one is interested in finding an efficient algorithm for solving a problem, where the notion of efficiency may involve a variety of (perhaps yet unknown) computing resources. Here though, we are concerned with the single resource time. The time requirements of an algorithm are usually expressed in terms of a single variable, the size of a problem instance, which is intended to reflect the amount of input data needed to describe the instance. This is intuitively appealing because one would expect the relative difficulty of problem instances to vary roughly with their size. The size of a problem instance is often measured in an informal way, so if time requirements are to be compared in a

precise way, care must be taken to define instance size in a uniform manner.

The description of a problem instance provided as input to a computer can be viewed as a single finite string of symbols, chosen from a finite fixed alphabet. Although there are many different ways in which instances of a given problem might be described, it is assumed that each problem has associated with it a fixed encoding scheme E which maps problem instances into the strings describing them. The input length for an instance I of a problem π is defined to be the number of symbols n in the description E(I) obtained from the encoding scheme for π, and it is this number n that is used as the formal measure of instance size.

The time complexity for an algorithm, defined as the largest amount of time needed by the algorithm to solve for each input length $|E(I)|$ an instance I of that length, thus depends upon the particular encoding scheme chosen. However, the standard encoding schemes used in practice always seem to differ at most polynomially from one another, so that any algorithm having polynomial time complexity under one of these encodings will have polynomial complexity under all the others. As it is difficult to imagine an encoding scheme for a naturally occurring problem that differs more than polynomially from the standard ones, there has been general agreement as to what constitutes a reasonable encoding scheme.

Although what is meant by reasonable cannot be completely (satisfactorily) formalized, the generally accepted meaning includes the notions of conciseness and decodability. The intent of conciseness is that instances of a problem should be described with the natural brevity that would be used in actually specifying those instances for a computer, without any unnatural padding of the input, as such padding might expand the input length so drastically that an exponential time algorithm would be artificially converted to an algorithm with only polynomial complexity. The intent of decodability is that, given any particular component of an arbitrary instance, a polynomial time algorithm can be specified for decoding a description of that component from any given encoded instance. In other words, for a problem to be considered realistically defined (encoded), the solution which satisfies the problem parameter values specified in each instance must itself not be so extensive that it cannot be described with an expression having length bounded by a polynomial function of the input length.

A naturally occurring (ie. the diagonal process is not employed) example which comes very close to the kind of function Theorem 1 predicts is the determination of the characteristic sequence X_{A_*} of the set of busy beaver numbers first studied by [Rado 1962ab]. By modifying the idea of what constitutes an acceptable program for computing the initial segments of this sequence, the desired example

of a discontinuous program-time trade is obtained. It is shown in [Daley 1976] that the set A_* which is defined by

$$n \varepsilon A_* \Leftrightarrow \exists m.n = \max \{\Phi_i(1) \mid \varphi_i(1)\downarrow \text{ and } \mu(i) \leq m\}$$

[where $\Phi_i(1)$ is the computation time of program i on initial tape input 1, $\varphi_i(1)\downarrow$ means that program i (in the standard numbering for Turing machines) on input 1 is defined (halts) and $\mu(i)$ is the program-size measure for $\{\varphi_i\}$, the length of the standard binary encoding of the integer i] has initial characteristic sequence segments x^n computable in time $O(n\log_2 n)$ by very short programs $K_\psi(x^n \mid n)$, i.e. with extremely low minimal-program complexity. In fact $K_\psi(x^n \mid n)$ grows more slowly as a function of n than every partial recursive function (equivalently, the running time as a function of m grows faster almost everywhere than every partial recursive function). It is also shown there is an algorithm which given an initial segment x^n finds a minimal-program (to within a constant) for x^n.

Intuitively, we can compute $X_{A_*}(m)$ for all $m \leq n$, given n, if we can determine all members of A_* which are $\leq n$. Corresponding to each member of A_* there is a program whose running time is that member. Thus to compute all members $\leq n$ of A_* we need only encode into a program all the programs corresponding to the members $\leq n$ of A_*. Both the size and number of these programs must, as a function of n,

grow more slowly than every partial recursive function. Furthermore, by their very nature the running times of all these programs are less than n. Thus we see that A_* consists of numbers which admit very short descriptions and which are easily obtainable from these descriptions.

However, these programs $K_\psi(x^n|n)$ do not constitute a reasonable encoding scheme for computing the characteristic sequence of the set of busy beaver numbers. This is evident, since X_{A_*} (the solution) grows in length directly with n by definition while $K_\psi(x^n|n)$ grows arbitrarily slowly as a function of n, the initial segments x^n of X_{A_*} of length n are not describable in length (or time) polynomially related to input length $K_\psi(x^n|n)$. So while the programs $K_\psi(x^n|n)$ are certainly concise by their very nature, they are not in general decodable. Thus for encoding-independent uniformity with other results and concepts in complexity theory, the program for determining X_{A_*} must include for each problem instance a technique for describing the desired solution within the polynomial limitation. In general, such a technique must be capable of describing place values in a sequence of length n, and therefore must in essence encode i for all $i \leq n$. To compute x^n and describe the result by any such program P, its length |P| must therefore grow more rapidly than every partial recursive function of $K_\psi(x^n|n)$ as n grows.

The related programs for computing $N_e(n)$, defined as the number of n-state binary Turing machines that halt, do not grow unreasonably fast but have running times which grow faster than any computable function of n. In particular [Rado 1962ab] observed that

$$1 < N_e(n) < N(n) = [4(n+1)]^{2n}$$

where N(n) is the number of all possible n-state (binary) Turing machines*. If N(s,n) denotes the number of n-state Turing machines which halt after exactly s shifts, the computation can be readily programmed as follows. Informally, one finds the value of N(s,n) by running each one of the n-state binary Turing machines, whose number is N(n), persisting through not more than the given number s of shifts and noting the number of those that stop after exactly s shifts. Thus

$$G(s,n) = \sum_{i=1}^{s} N(i,n)$$

is the number of n-state binary Turing machines that stop after not more than s shifts, so

*i.e., the number of distinct instructions governing overprint (2), shift (2) and state transfer (n states plus halt) for 0's and 1's is $[4(n+1)]^2$ for n possible states

$$G(s,n) \leq N_e(n).$$

The smallest value of s for which $G(s,n) = N_e(n)$ is $s = t$, where $t \in A_*$ is defined as the largest running time among all n-state binary Turing machines. Since program-size $\mu(i)$ is at most linearly related to number of states n, it follows from the previous example that t grows faster than every partial recursive function, and so obviously does tN(n), the running time of $N_e(n)$.

It should be noted that the set A_* of busy beaver numbers is retraceable [Rogers 1967] as shown in [Daley 1976], and it is evidently this property that leads to the discontinuous trade between program-size and computing time in the above examples.

Definition (Dekker, Myhill, Tennenbaum). A is retraceable if ($\exists$ partial recursive ψ) $\forall x$ [$x \in A \Rightarrow$ [$\psi(x)$ is defined; $\psi(x) = x$ if x is the smallest member of A; and $\psi(x)$ = the next smaller member of A if x is not the smallest member of A]].

Other examples of retraceable sets (trees of numbers) have similarly been proposed which can have short programs, but which are arbitrarily difficult to compute [Chaitin 1970]. Thus retraceability seems to be one property that can lead to the kind of

result in Theorem 1. That there are other properties that can similarly lead to the result is seen below.

The question whether for a function f with r-speedup (section 4.2) there must exist a recursive function which bounds the size of program necessary to effect the speedup was originally posed in [Blum 1971]. A negative answer for effective operators slightly larger than r appears in [Helm-Young 1971] where it is shown that functions f exist which have the property that if we are given any program P for computing the function and want to pass to a program P' which computes the function much more efficiently, then we can only do so at the expense of obtaining a much larger program. In fact, the function which describes the necessary increase in the size of the more efficient program P' must grow more rapidly than any recursive function. The functions f have speedup, but not only can one not effectively find the programs P' which admit the speedup, even if one could, their complexity must increase in such a way that their size becomes totally unrealistic. It is thus evident that the speedup property is also related to Theorem 1 in the sense that certain speedable functions [Blum-Marques 1973] exhibit the discontinuous trade between their program-size and computation time predicted by the theorem. That the programs for such functions are non-effective is of course already implied by Theorem 1. The connection between speedable functions and Theorem 1 is explored further in section 4.2.

Theorem 1 also suggests the following interpretation. It is very likely there are infinite sequences of finite functions f, which can be solved in polynomial time τK_ψ only by distributing the inputs among an ever larger (unbounded) number γK_ψ of hardware units, each accepting an amount of information $K_{\psi_i}(f) \leq K_\psi(f)$, the total information content of f. The model is then equivalent to the γK_ψ-state nondeterministic computation of f, using time polynomial in K_ψ, where f cannot be solved in time less than polynomial in γK_ψ deterministically. (A similar interpretation holds for Theorem 1, where τ is any time complexity class.)

In particular, if running time τ must be no larger than polynomial in $|P|$, then by Theorem 1 $\exists f \in M_n$ which requires $|P| > \gamma(K_\psi)$ for any recursive γ. This is depicted in Figure 2, where program P input is parsed into subprograms each having length $K_{\psi_i} \leq K_\psi$. Now each K_{ψ_i} is the shortest program for some partial computation of f which runs in time polynomial in K_ψ, due to the assumption that τ is polynomial in $|P|$ and ψ is a deterministic (universal) Turing machine. Assuming that the solution to f is describable in length polynomial in K_ψ, the computation is equivalent to a nondeterministic search by the pieces K_{ψ_i} for the relevant solution parts to f.

If all K_{ψ_i} subprograms output independent descriptive solution data having finite length, the total solution length would exceed a polynomial in K_ψ. Thus the role of the guessing module (Figure 3)

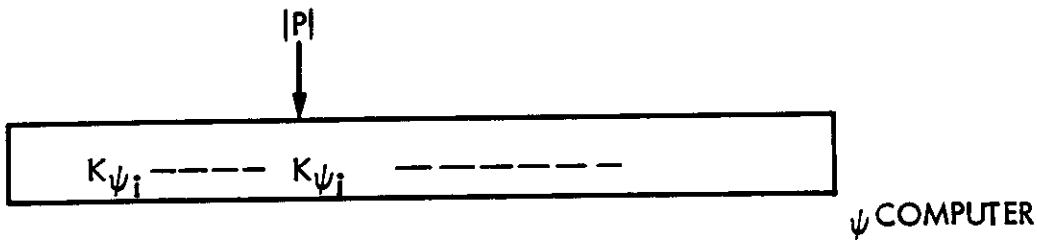


Figure 2. deterministic computation of f

is to select the length of each K_{ψ_i} and to simultaneously center attention on just those K_{ψ_i} that produce partial output (whose total output is a solution to f). Since f can be programmed in length K_ψ, the collection of K_{ψ_i} producing partial output will by definition be K_ψ in length (to within a constant). The collection will therefore consist of a finite number of subprograms K_{ψ_i} each running in time polynomial in K_ψ, and therefore collectively in time polynomial in γK_ψ nondeterministically. Related observations appear in [Berman-Hartmanis 1977, Kannan 1981].

Clearly an exciting result would be to discover a natural instance of such a problem. The search for such a sequence of f would

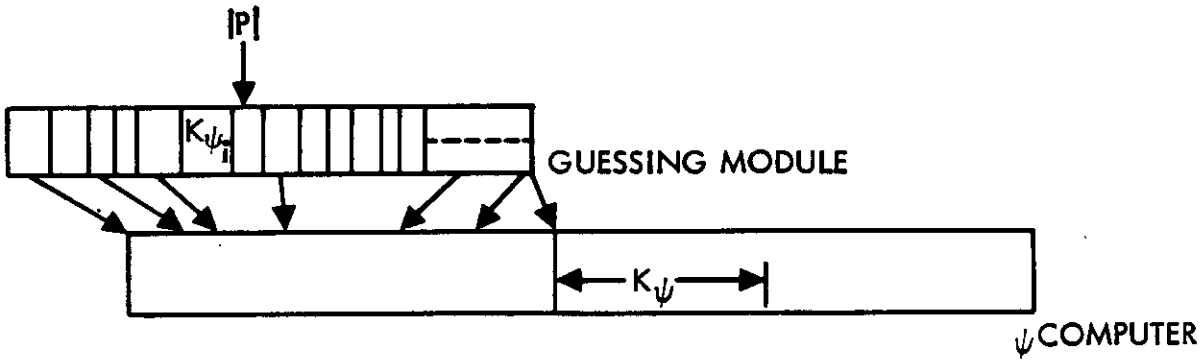


Figure 3. γK_ψ-state nondeterministic computation of f: guessing module prints all solution outputs from K_{ψ_i} parts (for all i) on tape to left of program K_ψ; K_ψ then "checks" the answer.

evidently lie at the extreme end of a progression of sets with ever diminishing input information rate, namely those of minimum accessible information rate [Yao 1982]. This provides an intuition that complexity classes defined in terms of information that can be accessed through a feasible computation may differ sharply from those associated with certain traditional encoding schemes. In a sense there is of course a practical preference for measures of accessible information in terms of K_ψ. On the other hand K_ψ is not effectively computable, whereas standard encoding schemes E for f can easily be defined and $|E(f)|$ effectively determined.

It is well known [Hartmanis 1978] that any proof procedure for a mathematical theory (e.g. Peano arithmetic) will have easily recognizable infinite sets T of trivially true theorems such that the length of their shortest proofs in the given formalism will grow faster than any given recursive function (of the length n of the theorems to be proved), and such sets can be effectively found. Even in the first order theory of addition of natural numbers (Pressburger arithmetic), there exist infinite sets T of theorems which are recognizable as true in polynomial time by some Turing machine φ, but φ uses at least (nondeterministic) time

$$T(n) \geq 2^{2^{qn}} \qquad q>0$$

for every element of T [Fischer–Rabin 1974]. It is evident from these remarks that our f-sequence in some sense lies at the opposite end of the spectrum from T. Thus f may be thought of as an infinite set γK_{ψ_i} of easy theorems K_{ψ_i} that takes arbitrarily long to recognize, but each member of f can be proved in polynomial time.

3. APPLICATIONS

3.1 Self-Modifying Programs

Even though all finite functions are necessarily recursive, not all finite functions are tractable - some require too much time to compute relative to their size. [Others require too large a program (as is the case with the finite subfunctions of a random function) relative to their size, so that most of the computation is essentially table lookup, i.e. there is little actual computation involved.] Self-modifying programs have been proposed as a way out of this dilemma. For example, presuming that the time limit for tractable computations is polynomial, suppose that f is a recursive function whose computation time only infrequently exceeds the polynomial limit. Given some program P for f, if P encounters one of those exceptional values, P can finish the computation and modify itself (thereby obtaining a new program P') so as to remember that difficult value and compute it the next time by table look-up. Through a formal argument it can be shown that every (possible infinite) recursive sequence is learnable in a certain sense (and so are some nonrecursive recursively enumerable sequences, but not all) [Schnorr-Fuchs 1975].

It is a result of Theorem 1 that one is unable in general to bound the length such new learned programs will require. This remains valid even when we relax our programming requirements so that it need only be possible to reconstruct reasonable approximations on the exceptional values encoded. Furthermore, Theorem 3 (Corollary 1) and Theorem 4 imply that the density of computations exceeding polynomial limits and thus requiring program modification may be so great that the time required for modifying is not feasible.

3.2 Probabilistic Algorithms

In the analysis of algorithms (programs) it is customary to draw a distinction between the worst case and expected time behavior of an algorithm [Karp 1976, Garey-Johnson 1979]. This distinction is prompted by the fact that for certain problems, while an algorithm may require an inordinately long time to arrive at a solution for the least favorable instances of the problem, on the average the required time is appreciably shorter. From a practical point of view, when many instances of a problem have to be solved, the average behavior is the more significant measurement of an algorithm.

This approach presupposes the existence of a known probability distribution on the space of all instances of the problem in question. Thus, for example, expected time of sorting algorithms is studied under the assumption that all n! permutations of the n

numbers to be sorted are equally likely. For many important computational problems we are on shaky ground in assuming a particular distribution on the space of instances of a problem. The relative frequency of the instance of the problem may be changing with time in an unpredictable and unmanageable way. The sample of instances actually appearing in a given application is often statistically biased in a manner not conforming to the assumptions made in the analysis of our algorithm.

An interesting recent innovation in algorithm design is the inclusion of stochastic moves and the allowance for an ε probability of error [Rabin 1976]. This represents a radically different approach and methodology for the study of probabilistic algorithms. Nothing is assumed about the distribution of the instances of the problem to be solved. Instead, randomness is incorporated directly into the algorithm. For an instance I of size n the randomness may take the form of choosing at random an integer $1 \leq b < n$, or chosing at random m integers $b_1, b_2, \ldots, b_m$ all smaller than n. An algorithm is then constructed involving a random step r so that for every instance I of the problem the expected computation time will be brief. In this approach randomness is not in the occurrence of the instances I, but is introduced into the algorithm itself.

Let Π be a computational problem, i.e., a collection of computational tasks I each of which is called an instance of Π. For

example, the sorting problem is the collecton of all n-tuples (for all n) of real numbers $I = (x_1, \ldots, x_n)$. For a given instance I the task is to produce a permutation ρ such that $x_{\rho(1)} \leq \ldots \leq x_{\rho(n)}$. When studying the complexity of a problem Π we associate with the instances $I \in \Pi$ a size $|I|$. The choice of the particular measure used to define size is not unique, and usually depends on the particular aspect of complexity under investigation. Generally speaking, the size $|I|$ is related to the amount of memory space used to store the input (the quantity of information required to specify I) or to natural parameters of specific problems, e.g., the number of vertices in a graph. A probabilistic algorithm for Π is defined as follows:

> Let $I \in \Pi$ be an instance of Π, $|I| = n$. At certain junctures in the solution of I randomly choose a number $1 \leq b < n$. Denote by b_i the i^{th} number chosen and denote by $r = (b_1, \ldots, b_m)$ the sequence. With the exception of the act of choosing the b_i, the algorithm proceeds completely deterministically. For simplicity it is assumed that all sequences r are equally likely.

An algorithm solves Π in expected time f(n) if for every $I \in \Pi$ such that $|I| = n$ the algorithm solves I in expected time $\leq f(n)$. Expected time is defined as the average of all solution times of I by the algorithm for all possible choice sequences r. A

probabilistic algorithm which for some instances $I \varepsilon \Pi$ and choices r may produce an incorrect solution is said to solve Π with confidence greater than $1-\varepsilon$ if for every $I \varepsilon \Pi$ the probability that it will produce an incorrect solution is smaller than ε, for $\varepsilon > 0$. In other words, for every instance I the relative frequency of the choice sequences r that lead to an incorrect solution is smaller than ε.

In the foregoing version of a probabilistic algorithm it has been tacitly assumed that the algorithm always terminates for every choice r. However, within the spirit of probabilistic computing, the definition can be extended to allow the sequences of choices $r = (b_1, b_2, \ldots)$ to occasionally become infinite. If the probability $p(I,m)$ of r becoming longer than m tends to zero rapidly enough, then the computation will terminate with probability 1 and still have short expected time. Other modifications and refinements are possible, again portraying the intuitive concept of probabilistic computation. For example, random choices from domains other than the set of natural numbers may be allowed, or one might consider different probability distributions on the choices [Chaitin-Schwartz 1978].

In view of the area-time exchange in VLSI the question naturally arises as to the possibility of saving a substantial amount of wire area or time by deliberately allowing some imprecision in

computation procedures. In other words, if each on-chip processor is given a random number generator, can the value of an arbitrary function f be determined with much less information exchanged? Such a computational model is then equivalent to implementing a parallel probabilistic algorithm. [Yao 1979] shows that the probabilistic 2-way communication complexity (i.e., the expected number of bits transferred between two processors cooperatively computing an arbitrary boolean function f on the worst input) decreases at most logarithmically over the deterministic 1-way communication model when $0<\varepsilon<{}_1/_2$. (The result in [Simon 1981] relating probabilistic and deterministic tape complexities of Turing machines limits the power of probabilistic algorithms for a single processor.) Theorem 2, which always requires at least an approximate solution as opposed to allowing an arbitrarily incorrect solution less than half the time ($0<\varepsilon<{}_1/_2$), shows there are computations f for which the VLSI complexity AT^2 or A^2T exceeds $\Omega[h(f)]$ or $\Omega[h(K_\psi(f))]$ for any recursive h. The reduction in VLSI complexity that can be achieved using parallel probabilistic algorithms is therefore clearly inadequate for those f having an h greater than exponential. Furthermore, Theorem 3 (Corollary 1) and Theorem 4 strongly imply that the density of computations f having such an h may be quite high, limiting the effective use of the probabilistic approach in reducing VLSI complexities.

3.3 Approximation Circuits

The notion of a boolean function may be generalized to that of a partially specified boolean function, whose value need not be specified for every combination of values of its arguments. A partially specified n-function may be described by a table with 2^n entries, in which X's (representing unspecified boolean values) are allowed as well as 1's and 0's.

There is an intimate connection between rate of transmission over a channel and the (circuit) complexity of approximate computation. Circuit-size lower bounds deal with maximum complexity over a class of functions (with a firm bound on the allowed error) while the related problem deals with average transmission and distortion rates. These averages are taken over both the distribution of the input and over the (probabilistic) transformation of input into output. For example, a typical procedure in rate-distortion theory regards a source sequence as a string of symbols, parses this string into substrings, and separately encodes each substring for transmission over the channel. An analagous procedure for complexity theory regards the table of a function as a string of symbols which can be parsed into substrings corresponding to subfunctions of the given function. These sub-functions can then be used to construct efficient circuits.

In conventional rate-distortion theory, the source string is divided into fixed-length blocks for encoding. If the source is ergodic, the sequence statistics for these blocks will approximate the ensemble statistics for the source. In this way, blocks whose statistics are similar to those of the source may be encoded efficently by random coding, while blocks whose statistics are much different from those of the source occur so infrequently that they can be dealt with by brute force (that is, encoded very inefficiently) without significantly degrading the average transmission or distortion rate. An adaptation of rate-distortion theory that does not rely on ergodicity (which ensures that fluctuations in blocks of moderate length are small), but rather on the hypothesis that after a certain prescribed number of symbols (possibly an excessive number) all fluctuations will have evened out, divides the source string into variable-length blocks, all of which can be encoded efficiently by random (circuit) covering. In effect, such an adaptation allocates quotas of complexity and distortion among the subfunctions of a given function. Thus, averaging over the input is replaced by an apportionment, while averaging over the transformation of input into output is replaced by random covering.

Utilizing these underlying connections to information theory, [Pippenger 1977] proved the following result: the complexity (ie., number of gates) of approximately computing a partially specified

n-argument boolean function in whose table a fraction d of the entries are unspecified and a fraction p of the specified entries are 1's with errors allowed in a fraction not more than e of the specified entries, is less by the factor $R(d,p,e) = (1-d)\,[H(p)-H(e)]$, where

$$H(z) = -z\log_2 z - (1-z)\log_2(1-z)$$

is the binary entropy function, than the complexity of exactly computing an arbitrary fully specified boolean function. Intuitively this factor is the minimum mutual information between a partially specified boolean random input variable F encoded for transmission over an arbitrary discrete memoryless channel with fully specified random variable G output, when F has the distribution

	X	d
F	1	p(1-d)
	0	(1-p)(1-d)

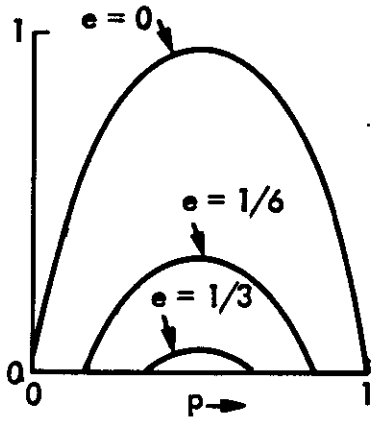
and the distortion $D(F,G) \leq e(1-d)$, where D is defined as the expectation of

		G		
		X	1	0
	X	0	0	0
F	1	∞	0	1
	0	∞	1	0

The expectation is of course ∞ if either of the ∞'s has positive probability. The behavior of the factor R(d,p,e) is shown in Figures 4 and 5 below.

This result holds with $0 \leq d < 1$, $0 < p < 1$ and $e < \min\{p, 1-p\}$. Note that if $e \geq \min\{p, 1-p\}$ every partially specified function can be approximated at sufficiently many (at least half) of its inputs by a constant function.

We are primarily interested in the behavior of R(d,p,e) at about $p = {}_1/_2$, since almost all binary input sequences (programs) of length 2^n will have about 2^{n-1} 1's, these representing the inputs of generally higher information content. Significant departures from $p = {}_1/_2$ imply input programs of low information. (It can be seen from Figure 4 that R(d,p,e) drops exponentially as p departs to either side of the value ${}_1/_2$.) Examination of Figures 6 and 7 reveals the difference in (worst case) circuit complexity for the approximate computation problem when the positions of the errors



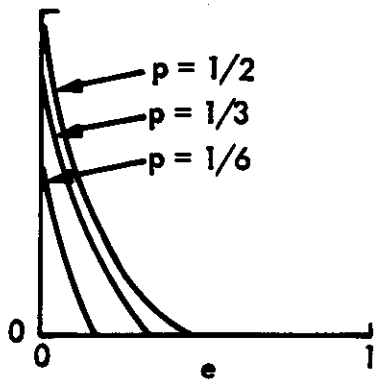
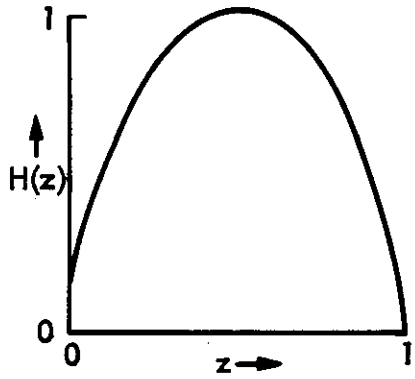


Figure 4. behavior of R(d,p,e) with d = 0 and e fixed

Figure 5. behavior of R(d,p,e) with d = 0 and p fixed



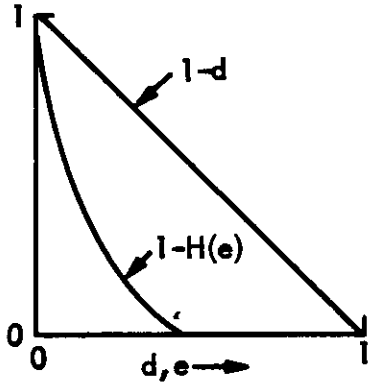


Figure 6. binary entropy function H

Figure 7. comparison of 1-d and 1-H(e)

are fixed rather than free. Thus circuit size drops linearly as (1-d) for fixed errors (unspecified input positions), while since 1-H(e) has slope $-\infty$ at e = 0, the first few errors allow a dramatic (exponential) drop in circuit complexity if their positions are free rather than fixed, and if p is near $_1/_2$.

That 1-H(e) drops to 0 at d, e = $_1/_2$ while 1-d does not reach 0 until d = 1, corresponds to the fact that any function can be approximated at one-half its inputs by a constant function, but an n-function specified at one-half its inputs might be an arbitrary (n-1) function.

Since replication of boolean approximation circuits leads to VLSI circuits for approximating binary functions, and as Theorem 2 allows some error in computing, there would seem to be limitations on what can really be achieved through such reduction factors R(d,p,e). In fact it is shown in [Pearl 1976] that

$$[H(p)-H(e)] = K_{\psi} + c$$

when $e<p\leq {}_1/_2$. From the proof of Theorem 2, it is seen that Theorem 1 implies the existence of planar circuit complexities P(f) for binary f which grow as γK_{ψ} for any recursive γ. The same holds for boolean functions f as a special case of Theorem 1. It is

evident that circuit reduction by factors close to the order γK_{ψ} as opposed to K_{ψ} are required in general to bring problems within feasible VLSI computing bounds.

4. FEASIBLE VLSI COMPUTATIONS AND SPEEDUP

4.1 Information Theoretic Parallel Computation Thesis

It is widely believed that the feasible sets (e.g., time = $n^{o(1)}$, space = $\log n^{o(1)}$) for sequential computers form a subclass of P (the set of problems computable in time polynomial in their input length). The belief that parallel algorithms which use a non-polynomial amount of hardware are as impractical as algorithms which require non-polynomial time, suggests that the feasible sets for parallel machines may also be a subclass of P, because polynomial hardware running for polynomial time can be simulated by a polynomial-time sequential (Turing) computation.

There has developed a substantial body of evidence showing that sequential Turing machine space can be simulated efficiently by parallel time - this relationship is called the parallel computation thesis. However, these simulations do not constrain the hardware used by the parallel machine, and in fact use an exponential (in the Turing machine space) amount of hardware. The thesis thus guarantees a fast parallelization if the original space is small, but does not bound the hardware required to achieve this in any way. Like Church's thesis, this thesis cannot be proved or disproved, because it relates an intuitive concept (parallel computation) to a mathematically precise one (Turing machines).

[Dymond-Cook 1980] have conjectured that no such bound on hardware is possible in general when transforming space to parallel time. However, they suggest that if the reversal used in the original Turing computation is small, then a fast parallel computation will use only polynomially more hardware than the original computation uses space (extended parallel computation thesis).

The following two theorems suggest an even further restriction on the subclass of P may be needed to characterize the feasible sets for parallel machines. Thus, beyond requiring both the space and reversal to be small (say polynomial in the logarithm of the input length), Theorem 6 requires that space and reversal under Turing computation be bounded in terms of (rate of growth of) the information K_ψ contained in the input.

Theorem 5. There are infinitely many finite functions f with polynomially bounded space and reversal such that parallel computation takes more hardware or time than any computable function of information content $K_\psi(f)$ if f runs in polynomial time, or f takes longer to compute than any recursive function of its program length |P|.

Proof. Sequential time τ (or total operation) is not less than linear in space or reversal, nor greater than linear in their

product [Hartmanis 1968]. By Theorem 1 there is an infinite sequence of finite functions which

1) run in polynomial time only by having programs longer than any computable function of information content $K_\psi(f)$ or

2) have short programs with arbitrarily large running times.

Here $\tau(|P|)$ in Theorem 1 is taken to be an arbitrary polynomial function. Then (1) implies that even when the space-reversal product is polynomial in $|P|$, the space (hardware) required for program input, or the reversal (parallel time [Hong 1980]) required to read the program on ψ, is not feasible when viewed in terms of $K_\psi(f)$. Likewise (2) implies that if the program for f is any shorter than in (1), the space - reversal product will become arbitrarily large in $|P|$, thus in $K_\psi(f)$, since $|P| \geq K_\psi(f)$ by definition. Therefore, even when the product of space and reversal is polynomially small, parallel computation of f will either take too long in terms of $K_\psi(f)$ or take too much hardware in terms of $K_\psi(f)$, or both for infinitely many f.

A more general result actually follows from Theorem 1, allowing arbitrarily small bounds on space and reversal. Of special interest

here would be those τ which are less than polynomial in the logarithm of input length.

Theorem 6. There are infinitely many finite functions f with τ-bounded space and reversal such that parallel computation takes unbounded hardware or time in terms of information content $K_{\psi}(f)$ if f runs in $\tau(|P|)$ time, τ any recursive function, or takes longer than any recursive function of input length $|P|$.

Proof. In Theorem 1, τ may be any pre-assigned recursive function, so the previous proof actually establishes the more general result stated here.

In the proof of the result in [Fine 1975] a message source M is reasonably compressible (i.e., has relatively short programs and running times) if and only if, for all $S \in M$, $K_{\psi}(S) < \gamma(c)$ where γ is any recursive-increasing function, and c is a constant which depends on τ, δ and ψ, but is independent of M. In other words, the low information-content problems S (here in the sense of program-size) are guaranteed to have programs of reasonable length for any fixed running time bound, and as such are efficient with respect to input (hardware) required. On the other hand, higher information-content problems require programs much longer than their information-content to run within the allowed time bound, and include many whose programs grow arbitrarily long. These programs

clearly contain large amounts of redundant information, which may be interpreted as multiple arrivals of certain of the inputs. Viewed in this manner, Theorems 1 and 6 imply that while all low information-content problems are guaranteed a feasible fast parallelization under the extended parallel computation thesis, and while some high information-content problems having low reversal (parallel time) with not too many multiple arrivals of input are similarly guaranteed, there are many finite functions f that, for a fixed bound on running time, require far too many multiple input arrivals to be simulated efficiently with parallel hardware.

Some results previously reported in the literature motivate the next theorem which provides additional intuition behind the extended parallel computation thesis:

1) [Hartmanis 1968] that for slowly growing ($\leq \log|P|$) reversal the speedup theorem does not hold for on-line Turing machine computations (there is linear speedup for all inputs if reversal grows faster than $|P|^{1+\varepsilon}$ for $\varepsilon > 0$, a result due to Blum strengthened to $\varepsilon|P|$ by [Fischer 1968])

2) [Fischer 1968] that off-line Turing machines always have linear speedup property for all reversal growth rates

3) [Lipton-Sedgewick 1980] that if inputs are allowed to arrive multiple times (free memory) the VLSI complexity of simple n-input, n-output functions remains unchanged, while the complexity of combining such simple functions may drop significantly

The problem of finding an F(n), for input of length n, such that for all nondecreasing functions growing faster than F(n) an on-line reversal speedup is possible and for all functions growing more slowly than F(n) no reversal reduction is possible was posed in [Fischer 1968]. [Hartmanis 1968] showed that when reversal R(n) is any constant function, or when R(n) is a certain function

$$R(n) = \min \{N \mid n \leq (G(N)+1)(2N+1)\}$$

(where G(N) is not too different from 2^N) which grows approximately as fast as $\log_2 n$, then no speedup is possible.

Theorem 7. An on-line Turing machine ψ has linear speedup property iff reversal $R(n) > \theta[\log_2 n+c]$ on input of length n for some fixed constant c independent of n.

Proof. Using a theorem due to Meyer [Loveland 1969], [Chaitin 1976b] has shown that an infinite binary string is recursive iff $\exists c \forall n$ the complexity $K_\psi(x^n)$ of its initial segments of length n

is $\leq c+\log_2 n$. Intuitively, an on-line reversal growth rate $\leq \theta[\log_2 n + c]$ thus implies that reversals are not being used generally for multiple (backup) looks at any fractional amount of input (i.e., εn for $0 < \varepsilon \leq 1$), while growth rates $>\theta[\log_2 n+c]$ clearly allow this possibility, and by analogy to off-line computation can be sped up by an arbitrary constant factor.

Intuitively then, the extended parallel computation thesis guarantees a practical fast VLSI parallelization only for Turing machine computations which cannot be sped up if inputs arrive only once, while if multiple arrivals are allowed the thesis guarantees a fast VLSI computation for linearly speedable computations with reversal $\leq \log_2 n+c$. Such computations may roughly correspond to complicated functions formed from simple functions through union, composition and/or alphabet change. The analogy between multiple-input arrival schedules in VLSI and off-line Turing machines was independently recognized in [Kedem-Zorat 1981].

4.2 K_ψ Characterization of Speedup

The interpretations in sections 2.4, 2.5 and 4.1 relating several of the present results to the speedup property indicate there may be a quantitative relationship involving K_ψ which characterizes the speedable or nonspeedable sets. Our notation will be primarily that of [Rogers 1967] and [Blum 1967].

Definition. Let φ_i be the i^{th} partial recursive function of one variable in a standard Godel numbering of p.r. functions. A family Φ_0, Φ_1, ... of functions of one variable is called a Blum measure on computation providing

1) domain (φ_i) = domain (Φ_i), and

2) the predicate $[\Phi_i(x) = m]$ is recursive in i, x and m.

Definition. An r.e. set A is speedable if for all i such that $W_i = A$ = domain φ_i and for all recursive functions h there exists j such that $W_j = A$, and

$$\exists^{\infty}x[\Phi_i(x)>h(x,\Phi_j(x))].$$

Furthermore, A is effectively speedable if j can be recursively computed from i and an index for h.

Intuitively, if A is speedable then for every program i for A and every recursive function h there is another program j for A which is an h-speedup of the first infinitely often, where j in an h-speedup of i on argument x if

$$\Phi_i(x)>h\ (x,\Phi_j(x)).$$

Definition. An r.e. set A is nonspeedable if there is a recursive h and an e such that $W_e = A$ and

$$\forall i\ [W_i = A \Rightarrow \Phi_e(x) \leq h(x,\Phi_i(x)) \text{ a.e.}].$$

The notation and basic assumption for relative recursiveness is the following. For any set X, φ_0^X, φ_1^X, φ_2^X, ... denotes a standard Godel numbering of all the partial recursive functions from integers into integers relativized to X. For all integers i and x, $\varphi_i^X(x)$ is a computation that proceeds algorithmically except that from time to time the computing agent may be required to obtain an answer to a question of the form "is n in X?". The answers to these questions are correctly and automatically supplied by some external device (usually referred to as an oracle for X), and obtaining an answer to one of these questions counts as a single step in the overall computation.

Definition. For each i, W_i^X = domain (φ_i^X). For any set X,

$$X' = \{x \mid \varphi_x^X(x) \text{ is defined}\} = \{x \mid x \in W_x^X\}$$

is called the jump or completion of X. Iterations of the jump are indicated by

$$X^{(0)} = X$$

$$X^{(n+1)} = \left[X^{(n)}\right]'$$

where $X^{(n)}$ is called the n^{th} jump of X.

Let F be the set of all total functions and predicates. We say two members of F are equivalent if each is recursive in the other, which defines an equivalence relation. The equivalence classes are called T-degrees of unsolvability, or Turing degrees. Roughly speaking, two functions or predicates are equivalent if they are equally difficult to calculate. Thus the degree of a function or predicate is a measure of the difficulty of calculating it. Let **a**,b, ... denote Turing degrees. For any degree **a**, **a**' is the uniquely determined degree of A', where A is any set in **a**. **a**' is called the jump or completion of **a**. **a** < b means that **a** is recursive in b and b is not recursive in a. 0 denotes the recursive degree, i.e., the degree of all recursive sets. It includes $\emptyset$ (the empty set) and ω (the well-ordered set of all integers) as members and is the smallest degree. 0' is the degree of the halting problem $\emptyset'$. A procedure is recursive in 0' (i.e., recursive in $\emptyset'$) if it is effective except for requiring the solution to certain individual and explicit instances of the halting problem, that is to questions that ask whether a certain specified and effectively recognizable kind of event occurs at

least once in the course of a certain effective computation. The intuitive significance of $\mathbf{a}'$ is parallel to that of $0'$. A procedure is recursive in $\mathbf{a}'$ (i.e., in A') if it is effective except for requiring the solution to individual instances of the (relativized) halting problem for procedures recursive in A, where A is some given set in $\mathbf{a}$.

Finally, we will need the following generalizations of Kolmogorov information to functions and sets:

Definition. The number of bits of information needed to specify a program p for the partial function φ relative to a set X of natural numbers is

$$K_{\psi}(\varphi|X) = \min\{\ell(p):\exists p\psi^{X}(p) = \varphi\}$$
$$= \infty \text{ otherwise.}$$

Definition. The information in a set A of numbers relative to another set X of numbers is defined by

$$K_{\psi}(A|X) = K_{\psi}(\lambda x[\text{if } x\varepsilon A \text{ then } 1 \text{ else } 0]|X)$$

where Church's lambda notation $\lambda x[-x-]$ is used to denote the partial function of x whose value is $[-x-]$.

Definition. The number of bits of information needed to specify a program p, relative to a set X of numbers, for the first n bits of the characteristic sequence for a set A of numbers is

$$K_\psi(A|X)^n = \min \{\ell(p):\exists p \psi^X(p) = \lambda x[\text{if } x\varepsilon A \text{ then 1 else 0}] \text{ for } 0 \leq x \leq n-1\}$$
$$= \infty \text{ otherwise.}$$

We use the following index set notation:

$$FIN = \{i|W_i \text{ is finite}\} = \overline{INF}$$

$$H_A = \{i|W_i \cap A \neq \emptyset\}$$

Theorem 8. If A is an r.e. set

a) A is speedable iff $\exists c \forall n[K_\psi(A'|\emptyset')^n > \log_2 n+c . K_\psi(\emptyset'|A')^n \leq \log_2 n+c]$

b) A is nonspeedable if $\exists c \forall n K_\psi(A'|\emptyset')^n \leq \log_2 n+c$

c) A is nonspeedable iff $\exists c \forall n K_\psi(H_{\overline{A}}|\emptyset')^n \leq \log_2 n+c$

Proof. Using a theorem due to A. Meyer, [Chaitin 1976b] has shown that an infinite binary string is recursive iff $\exists c \forall n$ the complexity $K_\psi(x^n)$ of its initial segments of length n is $\leq \log_2 n+c$.

Furthermore, an r.e. set $A \in \mathbf{a}$ is speedable iff $\mathbf{a}' > \mathbf{0}'$, and is nonspeedable iff H_A^- is recursive in $\emptyset'$ [Soare 1977]. And in [Marques 1975] it is shown there are nonspeedable sets in every r.e. degree, so in particular $\mathbf{a}' \leq \mathbf{0}'$ is sufficient for $A \in \mathbf{a}$ to be nonspeedable, but not necessary.

Results in [Young 1977] suggest both that the speedup phenomenon of Blum is highly complicated, and that its intuitive implications are not easily summarized. For example, by the standard speedup theorem, optimal programs may not exist. But even when optimal programs exist for a function, we may be unable to find them if we start with a program which is not provably equivalent to an optimal program. And for every program which is either optimal or nearly optimal, there is an effective procedure to find a nearly optimal program computing the same function which cannot be proved to be nearly optimal. In fact, some functions which do have (nearly) optimal computational methods, have no programs which compute the function which can be proved near optimal. Consequently, the existence of speedup among provably equivalent programs is independent of the function computed and depends only on the representation of the program defining the function. Related results appear in [Hartmanis-Hopcroft 1976, Hartmanis 1978, 1980].

4.3 1-Degrees and K_ψ

An iterated operation analogous in some respects to the jump operation can be defined which plays a role with respect to the 1-degrees (degrees of unsolvability with respect to one-one reducibility) similar to that of the jump operation with respect to the Turing degrees.

Definition. Let $\{\varphi_i : i\in\omega$ be an acceptable numbering of the partial recursive functions, and for every i let W_i be the domain of φ_i. For $n \geq 0$ let

$$A^{<0>} = A$$

$$A^{<1>} = \{i \mid W_i \wedge \bar{A}\neq\emptyset\} = H_{\bar{A}}$$

$$\vdots$$

$$A^{<n+1>} = \{i \mid W_i \wedge \overline{A^{<n>}}\neq\emptyset\} = H_{\overline{A^{<n>}}}$$

$A^{<n>}$ is called the n^{th} weak jump of A.

Definition. The set A is one-one reducible to B, which is written $A \leq_1 B$ iff there exists a one-one recursive function g, such that for all x

$$x\in A \iff g(x)\in B.$$

The set A is one-one equivalent to B, written $A \equiv_1 B$ iff $A \leq_1 B$ and $B \leq_1 A$.

The weak jump of a set is "weak" because it does not necessarily result in a set as high in the Turing degrees as the full jump of the set. The information an r.e. set A contains with respect to the Turing degrees can be measured by considering, for various values of n, the n^{th} jump of A. The set A is said to be low_n if $A^{(n)}$ has the same Turing degree as $\emptyset^{(n)}$, and $high_n$ if $A^{(n)}$ has the same Turing degree as $\emptyset^{(n+1)}$. The information content of an r.e. set can also be measured with respect to the 1-degrees by considering the various n^{th} weak jumps of A. Thus an r.e. set A is weak low_n if $A^{<n>} \equiv_1 \emptyset^{<n>}$ and weak $high_n$ if $A^{<n>} \equiv_1 \emptyset^{<n+1>}$. However, a set which is low_n with respect to 1-degrees is not necessarily low_n with respect to Turing degrees, and a set which is $high_n$ with respect to Turing degrees is not necessarily $high_n$ with respect to 1-degrees.

There is a strong connection between the computational complexity of a set and its information content as measured by various index sets. It is intuitively appealing to suppose that if one is given an appropriate measure of the information content of a computable set, then the more information the set contains the more difficult the set should be to compute. Some evidence exists that this intuitive notion is accurate.

In [Bennison 1979] it is shown that computable sets having fastest programs modulo some recursive function (nonspeedable, nonlevelable

and those with bounded complexity sequences) have low information content, while hard-to-compute sets (roughly those having an effective procedure for finding programs that are faster by any recursive amount than any program claiming to be fastest) have high levels of information-content as measured by various index sets.

In particular, it is shown that the non-speedable sets are exactly those r.e. sets A whose weak double jump $A^{<2>}$ has the same 1-degree as $\emptyset^{<2>}$, and that the effectively speedable sets are exactly those r.e. sets A whose weak jump $A^{<1>}$ has the same 1-degree as $\emptyset^{<2>}$. That the r.e. sets A for which $A^{<1>} \equiv_1 \emptyset^{<1>}$ (one-to-one equivalent to the halting problem) are the recursive sets, and those for which $A^{<0>} \equiv_1 \emptyset^{<1>}$ are the creative sets (i.e., r.e. complete), had previously appeared in the literature. By Theorem 8 we can therefore establish a connection between the algorithmic information measure K_ψ and information content with respect to the 1-degrees as measured by the index set FIN (the finite sets).

Theorem 9. If A is an r.e. set

$$\text{a)} \quad A^{<1>} \equiv_1 \text{FIN} \Rightarrow \exists c \forall n [K_\psi(A' \mid \emptyset')^n > \log_2 n + c.$$

$$K_\psi(\emptyset' \mid A')^n \leq \log_2 n + c]$$

b) $A^{<2>} \equiv_1 FIN$ if $\exists c \forall n K_\psi(A' | \emptyset')^n \leq \log_2 n+c$

c) $A^{<2>} \equiv_1 FIN$ iff $\exists c \forall n K_\psi (A^{<1>} | \emptyset')^n \leq \log_2 n+c$

Proof. It is shown in [Hartmanis-Lewis 1971] that for all n, $\emptyset^{(n)} \equiv_1 \emptyset^{<n>}$. So in particular $FIN \equiv_1 \emptyset^{<2>}$ where

$$\emptyset^{<2>} = \{i | W_i \wedge \overline{\emptyset^{(1)}} \neq \emptyset\} \text{ and}$$

$$FIN = \{i | W_i \text{ is finite}\}.$$

From [Bennison 1979] we know that the effectively speedable sets A are characterized by

$$A^{<1>} \equiv_1 \emptyset^{<2>}$$

and the nonspeedable sets A are characterized by

$$A^{<2>} \equiv_1 \emptyset^{<2>}$$

Thus the effectively speedable sets A are those where

$$A^{<1>} \equiv_1 FIN$$

while the non-speedable sets A are those where

$$A^{<2>} \equiv_1 FIN$$

Statements (b) and (c) now follow directly from Theorem 8, and since effectively speedable => speedable, so does statement (a).

Note that for A = ω

$$\omega^{<1>} = \{i \mid W_i \cap \overline{\omega} \neq \emptyset\} = \emptyset$$

$$\omega^{<2>} = \{i \mid W_i \cap \overline{\emptyset} \neq \emptyset\} = \emptyset^{<1>}$$

$$\omega^{<3>} = \{i \mid W_i \cap \overline{\emptyset^{<1>}} \neq \emptyset\} = \emptyset^{<2>}$$

$$\vdots$$

$$\omega^{<n>} = \{i \mid W_i \cap \overline{\emptyset^{<n-2>}} \neq \emptyset\}$$

$$= \emptyset^{<n-1>}$$

$$= \emptyset^{(n-1)}$$

so that both the effectively speedable as well as the nonspeedable A are evidently very different from ω.

It appears that the information a set contains with respect to the Turing degrees corresponds in a weaker fashion to the set's computational complexity than does its information content with respect to the 1-degrees. For example, if $A \leq_1 B$ then B

nonspeedable implies A nonspeedable, and A effectively speedable implies B effectively speedable, whereas A recursive in B yields neither implication.

The inequality $\deg(f) \leq \deg(g)$ for Turing degrees expresses the relative information between f and g (in the sense that if one knows g then one can compute f). As such, it is a qualitative sort of information. Algorithmic information content, on the other hand, measures quantitative differences between f and g. While a close connection has been hoped, no strong correlation between program-size and degrees of unsolvability other than 0 has been found. Theorem 8 shows there is a weak correlation between K_ψ and Turing reducibility among degrees. A related result was reported in the abstract [Chaitin 1972]. Theorem 9, however, implies there is a strong correlation between K_ψ and the 1-degrees, ie. between the algorithmic information in $A^{<n>}$ relative to $\emptyset^{(n)}$ and the information measured with respect to the 1-degrees.

Theorem 10. If A is an r.e. set different than ω, $A^{<n+1>} \equiv {}_1\emptyset^{<n+1>}$ if $\exists c \forall t K_\psi(A^{<n>} \upharpoonright \emptyset^{<n>})^t \leq \log_2 t+c$.

Proof. $\exists c \forall t K_\psi(A^{<n>} \upharpoonright \emptyset^{<n>})^t \leq \log_2 t+c$ implies $A^{<n>} \leq_T \emptyset^{<n>}$ by the Meyer-Chaitin result and $A^{<n+1>} \leq_T A^{<n>\prime} \leq {}_1\emptyset^{<n+1>}$ [Rogers 1967]. If A is an r.e. set different from ω then for all $n \geq 1$,

$\emptyset^{<n>} \leq {}_1A^{<n>} \leq {}_1\emptyset^{<n+1>}$ [Bennison 1979].

So we have $A^{<n+1>} \geq {}_1\emptyset^{<n+1>}$ and thus $A^{<n+1>} \equiv {}_1\emptyset^{<n+1>}$.

5. MEASURE INDEPENDENCE

A number of different measures of program size (descriptive) complexity have been introduced [Chaitin 1977]. All measures define the complexity of a finite string to be the length of a shortest program which prints the string. The measures differ in essentially two ways: the first is in the programming language used, and the second is that in some measures the programs are given additional information to help print the string. Unfortunately, these measures each have unique properties. The reasons for this are, in general, not well understood and there is disagreement as to which is most appropriate to use in different situations.

An example of one such property is oscillation. It has been shown that unrestricted descriptive complexity measures oscillate [Martin-Lof 1971]. Specifically, it can be shown that while most strings are of high (or maximal) complexity, arbitrarily deep complexity drops occur in their initial segments. These can be regarded as the results of partial computations in computing a function f. A related property is the length of a complexity oscillation, defined as the number of consecutive initial segments where complexities are low. It is easy to see that we can find arbitrarily long oscillations if we go far enough out. A sufficiently long run of 0's will do. The question is: are we guaranteed to find long oscillations relatively near the beginning

of the sequence? It can be shown that all sequences have long oscillations near the beginning. The idea behind this is fairly simple. Given x, recursively select from it a very sparse subsequence y. It is known that y has complexity oscillations. The original sequence x inherits these oscillations, and they now become the desired long oscillations due to the sparseness in our choice of y. It should be noted that this result holds for both absolute and conditional descriptive complexities of objects. Additional, somewhat surprising properties of various measures appear in [Kamae 1973, Chaitin 1976a].

We seek a unified approach to lower bounds for all the measures. As a step in this direction, it is noted that similarities between several complexity measures have been established. Specifically, equivalence can be shown among some pairs of measures to within an additive constant independent of the length of input string. For example, the conditional Chaitin complexity $H(x|n)$ and the conditional Kolmogorov-Solomonoff complexity measure $K_{\psi}(x|n)$ are roughly the same [Katseff-Sipser 1978]; the Schnorr process complexity $K^{P}(x)$ defined in terms of a universal directed process P (a more restrictive definition of complexity which does not oscillate) and the Kolmogorov complexity do not differ very much [Schnorr 1973, Schnorr-Fuchs 1975, Zvonkin-Levin 1970]. Likewise, the monotonic operator complexity [Levin 1973] which is similar in nature to the process complexity of Schnorr, nearly equals

$K_\psi(x)$. And the Loveland uniform measure K(x:n) almost equals $K_\psi(x|n)$ [Loveland 1969].

Closely related is the work of Shannon in which the complexity of a class of objects x (a random variable which can take on n values) is defined in terms of the probability distribution over that class. The individual complexity measures of Kolmogorov-Solomonoff and Chaitin have associated with them universal coding schemes. A universal encoding can be shown [Leung-Cover 1975] that has an expected codeword length with respect to any probability distribution on the set of possible outcomes, which is within a constant of the Shannon entropy H(x), interpreted as the minimal expected length of the description of x, establishing a tie between the individual complexity measure of Kolmogorov-Solomonoff and the average complexity measure of Shannon. The intuition behind the entropy H is so compelling that it would be disconcerting if H did not figure prominently in a description of the most efficient coding with respect to other less constrained coding schemes.

From these collective results, it appears that the connection between the descriptive complexity of a function f and the cost of computing f is a general one, independent to large degree of the particular descriptive measure employed to characterize f. In other words, the limitations on resources derived for various structures in computing a

function f remain essentially unchanged by substitution of a variety of information measures of the descriptive complexity of f.

In complexity theory one is concerned with categorizing functions or sets according to their relative difficulty of computation. The phrase "difficult to compute" may take on different meanings depending on which criteria (complexity theoretic properties) one uses to define what it means for a function or set to be hard to compute. In general, however, such criteria should yield the same classes of functions or sets regardless of the underlying complexity measure (in the sense of [Blum 1967]), e.g. tape, time, information, energy, etc. In other words, such criteria should be measure-independent.

For a property to have a recursion theoretic characterization it must be measure independent, for a recursion theoretic property is by its very nature measure-independent. It had been conjectured that there might exist some algorithm for determining whether or not a given complexity theoretic property is measure-independent. It is shown in [Bennison 1980] that measure-independence is undecidable, but that there is an effective procedure for finding a recursion theoretic characterization for any complexity theoretic property that is known in advance to be measure-independent.

6. SOFTWARE METRICS AND VLSI LOWER BOUNDS

One of the major concerns in the construction of software computational structures is the controlled introduction of complexity. The design of software systems has typically been based on purely qualitative guidelines, such as module independence or information hiding. However, recent interest in software quality assurance has motivated research efforts to develop and validate quantitative metrics to measure the complexity of software computational structures. If the measurement can be made from the specifications generated during the design phase, the system designer could use the metric evaluation in selecting between alternative designs or in altering poorly structured system components before a sizeable investment is made in the implementation of these components. In addition, quantitative metrics permit tradeoffs in the allocation of critical project resources, e.g. scheduling and cost vs. quality.

Much of the recent work in software engineering has explicitly recognized a fundamental relationship between complexity and software quality. Reducing costs and increasing quality are compatible goals that can be achieved when the complexity of the software is properly controlled. For example, the use of structured design methodologies allow the controlled introduction of complexity through levels of abstraction, virtual machines or layered hierarchies. By establishing an ordered discipline during the

design phase, these techniques have had significant impact on the production of higher quality, lower cost software. The common principle shared by these design methodologies is the careful structuring of the connections among the components of a system.

Research in software metrics can generally be divided into two categories: lexical metrics and connectivity metrics. The lexical metrics focus on the individual system components (subprograms, modules and procedures) and require a detailed knowledge of their internal mechanisms. Examples are metrics derived from counts of operators and operands, graph measures related to the count of the number of branch points in a program, and measures of software reliability factors based on the occurrence of various statement types. The common principle in these methods is the counting of lexical tokens without specific regard for the structure created by those tokens. The lexical measures are easy to calculate, and have been surprisingly robust in evaluating reliability aspects of computational structures comprised largely of independent modules.

Connectivity metrics, on the other hand, attempt to measure the degree of interaction between components of amodular computational structures. Examples of this type of metric are inter-level metrics [Yin-Winchester 1978], which observe the information flow across major levels in large hierarchic structures; semantic entropy measures [Alexander 1974], based on the idea that the connection between components is determined by their shared assumptions; the

partitioning formula of [Belady-Evangelisti 1981] derived from circuit clustering considerations in laying out chip designs; and the fan-in/fan-out technique of [Kafura-Henry 1981] which observes all communication patterns within a structure, rather than just those across level boundaries.

One of central problems in VLSI is the determination of the minimum amount of area required to lay out a network on a chip. In VLSI chips the computation is distributed over the chip, and the various processing elements must communicate via wires. These wires generally occupy more space than the computational elements themselves, and have become the principal factor in determining cost and performance. The relationship between communication requirements and complexity of computational structures has received much attention lately. Information transfer can be regarded as a measure of inherent modularity: a computation is inherently amodular if any way of partitioning the computation demands highly interacting parts. This amodularity entails tradeoffs among critical resources such as chip area, computation time and energy dissipation in circuits.

A number of metrics have appeared in the literature for measuring VLSI layout complexity. The bisection-width technique initially developed in [Thompson 1979] observes the necessary flow of information between two sides of an arbitrary partition of a circuit into nearly equal parts. [Hong-Kung 1981] on the other hand obtain

lower bound results corresponding to the limitations due to the I/O requirements of VLSI circuits, based on the computation graphs of specific operations. [Baudet 1981] accounts only for the memory required by a circuit in order to encode the input that has already been read before the output has been released completely, while [Brent-Kung 1980] account for memory as well as information transfer. [Vuillemin 1980]'s measure accounts for information transfer and the period of computations in pipelined chips. [Chazelle - Monier 1981]'s metric assumes the time for propagating information is linearly proportional to distance, rather than a constant, and accounts for information transfer. They also consider a more restrictive metric tailored to NMOS technology which additionally assumes the current density of electrical power supplied through wires to a circuit is bounded by a constant. [Leighton 1981] uses crossing number and wire area arguments to find lower bounds on layout area and maximum edge length (length of the longest wire in any layout of a network on a chip).

A fair degree of similarity appears between the software metrics and the VLSI measures. Whereas the development of software metrics was primarily driven by reliability concerns, the driving force behind the VLSI measures was interest in determining lower bounds on the difficulty of parallel computation. Under appropriate mappings [Cuykendall 1982ab], the VLSI measures provide some degree of insight into assessing the validity of measures of software design

complexity on an absolute rather than relative scale [Chapin 1979], as shown by the following theorem.

Theorem 11. If f is a recursive function, then every lower bound for the VLSI measure AT^2 of f is also a lower bound for the software metric $[h\sum_p(mn)]^2$ of any program P which computes f, where procedures p of P have fan-in m, fan-out n and h is an upper bound on the run time of procedure code. Furthermore, $[h\sum_p(mn)]^2$ arises naturally as a measure of program interconnectivity.

Proof. If procedure p has fan-in m (the number of local flows into p plus the number of data structures from which p retrieves information) and fan-out n (the number of local flows emanating from p plus the number of data structures which p updates), the number of distinct information paths connecting p to its environment is mn. The interconnections among components of P are its procedures. If f is computed by a VLSI circuit with bisection-width w and area A, then $A = \Omega(w^2)$. The bisection-width w is the minimum interconnection (ie, number of wires), for all possible VLSI circuits which implement programs P computing f, across the cut boundary dividing the circuit inputs equally. Each wire connects two gates in the implementation, one on either side of the cut.

Since each such wire represents an environmental path between components of the implemented P for at least one procedure $p \in P$,

$$\sum_p (mn) \geq w$$

$$[\sum_p (mn)]^2 = \Omega(w^2)$$

and every lower bound technique for the bisection-width of a graph is also a lower bound technique for both the software metric $[\sum_p(mn)]^2$ and layout area A. If the total amount of information that must flow across the bisection boundary during the entire computation is I, then the time taken must satisfy $T>I/w$. Thus a measure which arises naturally within this model is $AT^2 = \Omega(I^2)$. If h is an upper bound on the total time information is transferred among components of P, then every lower bound for the VLSI measure of difficulty AT^2 of f is also a lower bound for the software interconnectivity metric $[h\sum_p(mn)]^2$ which arises in a natural way from the VLSI implementation of P.

Both software and VLSI metrics permit evaluation of computational structure and useful trades in critical resources to be made early in the design phase. As the use of high-level languages [Hoare 1978, Rem, et al 1983, Snepscheut 1983] for designing chips increases, combined with the possibility of larger integration and bigger chips, the asymptotic properties of these metrics will become indistinguishable.

7. CONCLUSION

7.1 Open Problems

The results described in this work reveal some fundamental connections between Kolmogorov information and complexity theoretic properties of computation. These connections can have serious implications about the feasibility of large scale parallel computation.

It was shown in Theorem 3 that there is an infinite hierarchy of finite functions f of arbitrarily increasing degree of amodularity which cannot be approximated to within reasonable $\delta(f)$ by functions f' having an amodularity degree lower in the hierarchy. However, our theorem establishes only the sparse existence of functions which are increasingly amodular, namely one such f for approximately each $\gamma(n)$ with $n \geq n_0$. Thus, the distance between $\Sigma^{\gamma(n)}$ and $\Sigma^{\gamma(n+_1)}$ can be very large, and in fact becomes arbitrarily large with ever increasing γ. The high-density existence of such functions projected in Corollary 1, say one for each $\gamma(n_0)+k$, $k=_{0,1}$ ... (which would seem to have serious implications for very large scale computations such as weather prediction), depends on whether the Helm-Young Theorem extends to all sufficiently large operators R, as shown in Theorem 4. Their result is conjectured to hold for all sufficiently large operators on the basis that it should be more difficult to bound the size of programs effecting large speedups

than those bounding smaller speedups. A slightly stronger result appears in [Constable-Hartmanis 1971] and independently in [Meyer-Fischer 1972], the question of extension to all total effective operators R remaining open.

In view of the area-time exchange inherent in VLSI models of computation, the question naturally arises whether a physical computational process exists which is strictly more powerful than VLSI at quantum limits. If such a model can be defined and constructed, some of the limitations shown in our Theorems 1 through 6 can be avoided.

There has been much discussion in the literature recently about the physical limitations of the computation process and information transfer. For example, the mathematical existence of quantum mechanical models of Turing machines that dissipate no energy has been shown in [Benioff 1982]. For such models, limits on computation speed which normally arise from energy dissipation considerations are not present. One can in principle at least increase the computation speed by increasing the average system energy without introducing state degradation and energy dissipation. Whether or not such models can actually be physically constructed is, however, an open question.

7.2 New Directions

The modern theory of high-speed computation is based on the idea of a binary switch. In order to carry out a computation, individual bits of data are stored as switch positions, transmitted to interact with other switches storing data and the resulting switch positions stored. A very large number of these switching steps are required even in simple computations. In VLSI the role of the binary switch is carried out by the transistor, wires (substrate channels) are used to transmit stored data and gates perform the interactions.

The result of integrated circuit evolution is that we are approaching systems with significantly higher degrees of parallelism. As VLSI approaches element sizes at the quantum limit, circuit performance will depend primarily on internal transmission line delays due to length, resistance and capacitance considerations of interconnecting wires. This will be true as long as the computation carried out by the circuit requires transmission of causal information, and channel rates do not exceed the speed of light. That is, during the computation either signals are required which are intended to cause another event independent of the source event, or signals are required which are intended to carry information between two events having a common cause, where propagation of such signals is limited to the speed of light. Significantly, recent theoretical developments in quantum theory

predict that under some circumstances this limit is exceeded for signals of the second kind.

The traditional common-sense interpretation of nature is founded on three premises which are widely assumed to be self-evident. One is realism: that regularities in observed phenomenon are caused by some physical reality (i.e., properties) that exists independently of human observers. The second premise allows the free use of inductive inference, thus enabling valid conclusions to be drawn from consistent observations. The third premise is known as Einstein separability (or Einstein locality): that no influence of any kind can propagate faster than the speed of light. Any theory that incorporates them is called a local realistic theory. An argument derived from these premises leads to an explicit prediction of the outcome of a class of experiments in the physics of elementary particles. The experiments are concerned with correlations between distant events and with the causes of those correlations. Quantum theory can also be employed to calculate the results of these experiments. Surprisingly, the two predictions differ sharply, and so either the local realistic theories or quantum mechanics must be wrong [d'Espagnat 1978, Clauser - Shimony 1978].

In particular, local realistic theories predict a relation called the Bell inequality will be obeyed, whereas quantum mechanics

predicts a violation of the inequality [Bell 1964, Clauser - Horne 1974]. The relation, involving the spin of pairs of particles defined by components along three axes in space which need not be at right angles to one another, imposes a limit on the extent of correlation that can be expected when different spin components of mutually distant particles are measured, and this limit is expressed in the form of an inequality. Quantum theory predicts that for some choices of axes the limit will be exceeded. Recent experiments provide strong evidence that not only is the inequality violated, but it is violated in precisely the way predicted by quantum mechanics. Figure 8 shows the results of the proton experiment carried out in 1976 by M. Lamehi-Rachti and W. Mittig of the Saclay Nuclear Research Center in France. The negative correlation between the values of different spin components is given as a function of the angle between the settings of two spin-measuring instruments.

A well established but nonetheless surprising property of protons (and many other particles) is that no matter what axis is chosen for a measurement of a spin component, the result can take on only one of two values, for example + and -. A correlation of -1 would therefore indicate that the components invariably had opposite values. The Bell inequality states that the correlation at any angle must be on or above the diagonal line. Five experiments to date have resulted in violation of the inequality and in agreement with quantum mechanics. Four of these did not directly measure spin

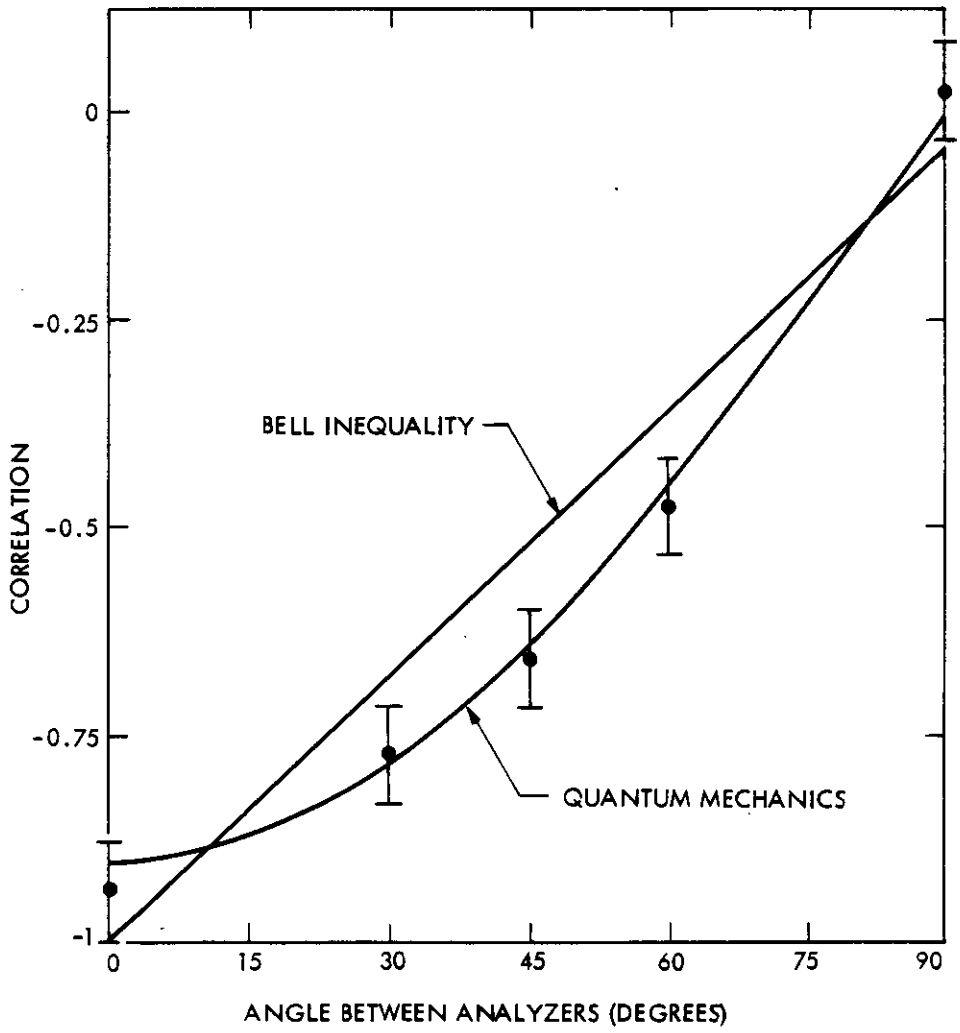


Figure 8. results of a distant-correlation experiment testing the foundations of microphysics

components, but instead measured the polarization of photons, which is the property of a photon that corresponds to the spin of a material particle. The violation of the Bell inequality implies that at least one of the three premises of local realistic theories must be in error. Einstein separability is considered the most plausible candidate.

The mechanism of the distant-correlation experiments leading to the inferred instantaneous linkages between events may be employed to define a thought experiment for testing whether complexity of computation based on violation of separability differs significantly from the VLSI model complexity at quantum limits [Cuykendall 1984b]. In such a computation the role of the binary switch is carried out by measuring the spin components of certain elementary particles along three arbitrary space axes A, B and C. [An equivalent model may be formulated by measuring polarization components of photons along any three directions in space, or by measuring any other three stable properties A, B and C of certain elementary particles or entities, each of which can take on only one of two values (e.g., + and -).] Whether or not such an experiment can actually be constructed and carried out remains an open and beckoning question.

REFERENCES

[1] H. Abelson, **Lower Bounds on Information Transfer in Distributed Computations**, JACM Vol. 27 No. 2, April 1980, pp. 384-392

[2] C. Alexander, **Notes on the Synthesis of Form**, Harvard Press, 1964

[3] D. A. Alton, **Diversity of Speed-Ups and Embeddability in Computational Complexity**, The Journal for Symbolic Logic, Vol. 41, Number 1, March 1976, pp. 199-214

[4] J. Backus, **Can Programming be Liberated From the von Neumann Style? A Functional Style and Its Algebra of Programs**, Comm ACM, Vol 21 No. 8, Aug 1978

[5] G. Baudet, **On the Area Required by VLSI Circuits**, Conf. VLSI Systems and Computations, CMU Oct. 1981

[6] L. A. Belady and C. J. Evangelisti, **System Partitioning and Its Measure**, J. of Systems and Software 2, 23-29 (1981)

[7] J. S. Bell, **On the Einstein Podolsky Rosen Paradox**, Physics Vol 1, No. 3, pp. 195-200, Nov-Dec 1964

[8] P. Benioff, **Quantum Mechanical Models of Turing Machines That Dissipate No Energy**, Phys. Rev. Lett.V.48, No. 23, pp. 1581-1585, 7 June 1982

[9] V. L. Bennison, **Information Content Characterizations of Complexity Theoretic Properties**, Lecture Notes in Computer Science, Vol. 67 (K. Weihrauch, Editor) Springer Verlag, Berlin, 1979, pp. 58-66

[10] _____ **Recursively Enumerable Complexity Sequences and Measure Independence**, J. Symbolic Logic, Vol 45, No. 3, Sept. 1980

[11] L. Berman and J. Hartmanis, **On Isomorphisms and Density of NP and Other Complete Sets**, SIAM J. Computing 6, 1977, pp. 305-322

[12] M. Blum, **A Machine Independent Theory of the Complexity of Recursive Functions**, J. ACM, Vol. 14, 1967, pp. 322-336

[13] _____ **On Effective Procedures for Speeding Up Algorithms**, J. ACM 18, 2, April, 1971, pp. 290-305

[14] M. Blum and I. Marques, **On Complexity Properties of Recursively Enumerable Sets**,, J. of Symbolic Logic, V-1 38, No. 4, Dec. 1973, pp. 579-593

[15] A. Borodin, **Computational Complexity and Existence of Complexity Gaps**, J. ACM, Vol 19 (1972), pp. 158-174

[16] _____ **On Relating Time And Space to Size and Depth**, SIAM J. Computing, Vol 6, No. 4, Dec 1977

[17] R. P. Brent and H. T. Kung, **The Chip Complexity of Binary Arithmetic,** 12th Annual ACM Symposium on Theory of Computing, 1980, pp. 90-120

[18] L. L. Campbell, **Entropy as a Measure,** IEEE Transactions on Information Theory, Vol IT-11, Jan 1965, pp. 112-114

[19] G. J. Chaitin, **On the Difficulty of Computations,** IEEE Transactions on Information Theory, Vol. IT-16, January 1970, pp. 5-9

[20] _____ **Information-Theoretic Aspects of the Turing Degrees,** Abstract 72T-E77 AMS Notices 19 (1972), pp. A-601, A-602

[21] _____ **Information-Theoretic Limitations of Formal Systems,** J. ACM 21, 3 (July 1974), pp. 403-423

[22] _____ **A Theory of Program Size Formally Identical to Information Theory,** JACM, Vol. 22, 1975

[23] _____ **Program Size, Oracles, and the Jump Operation,** IBM Research Report, 1976a

(24] _____ **Information-Theoretic Characterizations of Recursive Infinite Strings,** IBM Research Report RC 5819 (#25225), and Theoretical Computer Science 2, 45, 1976b

[25] _____ **Algorithmic Information Theory,** IBM J. Res. Develop., July 1977, pp. 350-359

[26] G. J. Chaitin and J. T. Schwartz, **A Note on Monte Carlo Primality Tests and Algorithmic Information Theory,** Communications on Pure and Applied Mathematics, Vol. XXXI, 1978, pp. 521-527

[27] N. Chapin, **A Measure of Software Complexity,** National Computer Conf. 1979, pp. 995-1002

[28] B. Chazelle and L. Monier, **A Model of Computation for VLSI with Related Complexity Results,** 13th Ann ACM Symp Theory of Computing, 1981, pp. 318-325

[29] J. F. Clauser and M. A. Horne, **Experimental Consequences of Objective Local Theories,** Physical Review D, Vol 10, No. 2, pp. 526-535, July 15, 1974

[30] J.F. Clauser and A. Shimony, **Bell's Theorem: Experimental Tests and Implications,** Reports on Progress in Physics, Vol 41, No. 12, pp. 1881-1927, Dec 1978

[31] R. Constable and J. Hartmanis, **Complexity of Formal Translations and Speedup Results,** Proceedings 3rd Annual ACM Symp., May 1971 pp. 244-250

[32] R. Cuykendall, **Kolmogorov Complexity and Performance Measurement,** Computer Science Dept. Research Seminar Notes, UCLA 1973

[33] _____ **Relations Among Software Interconnectivity Metrics, VLSI Lower Bounds and Programming Primitives,** Computing Memorandum No. 490, Jet Propulsion Laboratory, California Institute of Technology, Sept. 13, 1982a

[34] _____ **Software Metrics and Lower Bounds**, IEEE Computer Society Proceedings, Workshop on the Intersection of VLSI and Software Engineering, Oct 1982b

[35] R. Cuykendall et al, **VLSI Design Synthesis and Measurement**, IEEE Computer Society Proceedings, Workshop on the Intersection of VLSI and Software Engineering, Oct. 1982c, and in J. Systems and Software, to appear 1984a

[36] R. Cuykendall, **Quantum Mechanical Models and Computational Complexity**, Caltech/Jet Propulsion Laboratory Technical Report D-1641, 7 June 1984b

[37] R. P. Daley **Non-Complex Sequences: Characterizations and Examples**, J. Symb. Logic, 41, 1976.

[38] R. A. DeMillo and R. J. Lipton, **Some Connections Between Mathematical Logic and Complexity Theory**, Proc. 11th Ann ACM Symp. on Theory of Computing, Atlanta, GA, 1979, pp. 153-159

[39] P. E. Dymond and S. A. Cook, **Hardware Complexity and Parallel Computation**, IEEE FOCS Conference, 1980, pp. 360-372

[40] B. d'Espagnat, **Use of Inequalities for the Experimental Test of a General Conception of the Foundations of Microphysics**, Physical Review D, Vol. II, No. 6, pp. 1424-1435, Mar 15, 1975; Part 2, Vol. 18, No. 2, pp. 349-358, July 15, 1978

[41] T. Fine, **Uniformly Reasonable Source Encoding Is Often Practically Impossible**, IEEE Trans. on Information Theory, Vol. IT-21, No. 4, July 1975

[42] P. C. Fischer, **The Reduction of Tape Reversals for Off-Line One-Tape Turing Machines**, Journal of Computer and System Sciences, Vol. 2, No. 2, 1968, pp. 136-147

[43] M. J. Fischer and M. O. Rabin, **Super-Exponential Complexity of Presburger Arithmetic**, SIAM-AMS Proceedings, Vol. 7, 1974, pp. 27-41

[44] M. R. Garey and D. S. Johnson, **Computers and Intractability**, W.H. Freeman & Co., 1979

[45] W. M. Gentlemen, **Some Complexity Results for Matrix Computations on Parallel Processors**, JACM 25, 1 Jan 1978, pp. 112-115

[46] D. Yu Grigoriev, **An Application of Separability and Independence Notions for Proving Lower Bounds on Circuit Complexity**, Notes of Scientific Seminars, Steklov Math Inst., Leningrad, Vol. 60, 1976

[47] J. Hartmanis, **Tape-Reversal Bounded Turing Machine Computations**, Journal of Computer and System Sciences, Vol. 2, No. 2, 1968, pp. 117-135

[48] J. Hartmanis and F. E. Lewis, **The Use of Lists in the Study of Undecidable Problems in Automata Theory**, J. Computer and Systems Science 5, 1971, pp. 54-66

[49] J. Hartmanis and J. E. Hopcroft, **Independence Results in Computer Science,** ACM SIGACT at News 8, No. 4, 1976, pp. 13-24

[50] J. Hartmanis, **Relations Between Diagonalization, Proof Systems and Complexity Gaps,** Proceedings Ninth Annual ACM Symposium on Theory of Computing, 1977

[51] _____ **Feasible Computations and Provable Complexity Properties,** CBMS-NSF Regional Conf. Ser. in Appl. Math., SIAM, 1978

[52] _____ **On the Succintness of Different Representations of Languages,** SIAM J. Computing, Vol 9, No. 1, Feb 1980

[53] J. Helm and P. Young, **On Size vs Efficiency for Programs Admitting Speed-Ups,** J. Symbolic Logic, Vol 36, No. 1, March 1971

[54] F. C. Hennie, **One-Tape Off-Line Turing Machine Computations,** Information and Control 8, 1965, pp. 553-578

[55] _____ **On-Line Turing Machine Computations,** IEEE Transactions on Electronic Computers, Vol. EC-15, No. 1, February 1966, pp. 35-44

[56] C. A. R. Hoare, **Communicating Sequential Processes,** Comm ACM Vol. 21, No. 8, 1978, pp. 666-677

[57] J. W. Hong, **On Similarity and Duality of Computation,** 21st IEEE Annual Symp. on Foundations of Computer Science, 1980.

[58] J. W. Hong and H. T. Kung, **I/O Complexity: The Red-Blue Pebble Game,** 13th Ann. ACM Symp. Theory of Computing, 1981, pp. 326-333

[59] D. Kafura and S. Henry, **Software Quality Metrics Based on Interconnectivity,** J. Systems and Software 2, 121-131, 1981

[60] T. Kamae, **On Kolmogorov's Complexity and Information,** Osaka J. Math 10, 1973, pp. 305-307

[61] R. Kannan, **Circuit-Size Lower Bounds and Non-Reducibility to Sparse Sets,** MIT Laboratory for Computer Science TM-205, October 1981.

[62] R. M. Karp, **The Probabilistic Analysis of Some Combinatorial Search Algorithms,** in Algorithms and Complexity: New Directions and Recent Results, edited by J. F. Traub, Academic Press, 1976, pp. 1-19

[63] H. P. Katseff and M. Sipser, **Several Results in Program Size Complexity,** Dept EE and Computer Sciences, UC Berkeley, 1978

[64] Z. M. Kedem and A. Zorat, **On Relations Between Input and Communication/Computation in VLSI,** IEEE 22nd FOCS Conference, Nashville, TN, 1981

[65] A. N. Kolmogorov, **Three Approaches to the Quantitative Definition of Information,** [Russian] Problemy Peradachi Informatsii, Vol. 1., No. 1., 1965.

[66] F. T. Leighton, **New Lower Bound Techniques for VLSI,** FOCS, Nov. 1981

[67] S. K. Leung-Yan-Cheong and T. M. Cover, **Some Inequalities Between Shannon Entropy and Kolmogorov, Chaitin and Extension Complexities**, Tech. Rept. 16, Dept. of Statistics, Stanford Univ., 1975

[68] L. Levin, **On the Notion of Random Sequence**, Soviet Mathematics-Doklady, Vol. 14, 1973, pp. 1413-1416

[69] R. J. Lipton and R. E. Tarjan, **Applications of a Planar Separator Theorem**, SIAM J. Computing, Vol. 9, Number 3, August 1980, pp. 615-627

[70] R. J. Lipton, and Sedgewick R., **Lower Bounds for VLSI**, Proc. 13th Ann ACM Symp. Th. of Computing, May 1981 pp. 300-307

[71] D. Loveland, **A Variant of the Kolmogorov Concept of Complexity**, Information and Control, Vol. 15, 1969, pp. 510-526.

[72] G. Mago, **A Network of Microprocessors to Execute Reduction Languages**, Int'l J. Compr. & Info. Sci. 1979, Vol 8 No. 5 (Part I), Vol 8 No. 6 (Part II)

[73] I. Marques, **On Degrees of Unsolvability and Complexity Properties**, The Journal for Symbolic Logic, Vol. 40, Number 4, December 1975, pp. 529-540

[74] P. Martin-Lof, **Complexity Oscillations in Infinite Binary Sequences**, Z. Wahrscheinlichkeit Theorie, verw. Geb. 19, 1971 pp. 225-230

[75] C. A. Mead and M. Rem, **Cost and Performance of VLSI Computing Structures**, IEEE, SC-14, 1979

[76] A. Meyer and P. Fischer, **Computational Speedup By Effective Operators**, Journal of Symbolic Logic, Vol 37, No. 1, Mar 1972

[77] W. J. Paul, **Kolmogorov Complexity and Lower Bounds**, Second Int'l Conf. on Fundamentals of Computation Theory, pp. 325-334, Sept. 1979

[78] W. J. Paul, J. I. Seiferas and J. Simon, **An Information-Theoretic Approach to Time Bounds for On-line Computation**, Proc. 12th Annual ACM Symposium on Theory of Computing, pp. 357-367, 1980

[79] J. Pearl, **Theoretical Bounds on the Complexity of Inexact Computations**, IEEE Trans IT Vol. 22, No. 5, September 1976

[80] N. Pippenger, **Information Theory and the Complexity of Boolean Functions**, Math. Syst. Theory 10, 129-167 (1977)

[81] N. Pippenger and M. J. Fischer, **Relations Among Complexity Measures**, JACM, Vol. 26, No. 2, 1979

[82] M. O. Rabin, **Probabilistic Algorithms**, in Algorithms and Complexity: New Directions and Recent Results, edited by J. F. Traub, Academic Press, 1976, pp. 21-39

[83] T. Rado, **On Non-Computable Functions**, Bell System Technical Journal, May 1962a

[84] _____ **On a Simple Source for Non-Computable Functions,** Proceedings of the Symposium of Mathematical Theory of Automata (MRI Symposia Series, Vol. 12) Polytechnic Press, Brooklyn, 1962b, pp. 75-81

[85] S. Reisch and G. Schnitger, **Three Applications of Kolmogorov Complexity,** Proc. 23rd IEEE Symp. on Foundations of Computer Science, 1982, pp. 45-52

[86] M. Rem, J. van de Snepscheut and J. Udding, **Trace Theory and the Definition of Hierarchical Components,** Third Caltech Conference on VLSI, Pasadena, CA, 1983

[87] H. Rogers, **Theory of Recursive Functions and Effective Computability,** McGraw-Hill, 1967

[88] J. E. Savage, **On the Performance of Computing Systems,** Ctr for Information Sciences, Brown Univ., 1973

[89] J. E. Savage and S. Swamy, **Space-Time Tradeoffs for Oblivious Integer Multiplication,** Lecture Notes in Computer Science, Vol. 71, Berlin, 1979

[90] J. E. Savage, **Planar Circuit Complexity and The Performance of VLSI Algorithms,** CMU Conf. on VLSI Systems & Computations, October CMU 1981, pp. 61-68

[91] C. P Schnorr, **Process Complexity and Effective Random Tests,** Comp. and Syst. Sci., Vol. 7, 1973

[92] C. P. Schnorr and H. Fuchs, **General Random Sequences and the Concept of Learnable Sequences,** Preprint, Fachbereich Mathematik, Universitat Frankfort, 1975

[93] C. P. Schnorr, **The Network Complexity and the Turing Machine Complexity of Finite Functions,** Acta Informat 7, 1976, pp. 95-107

[94] J.R. Shoenfield, **Mathematical Logic,** Addison-Wesley, 1967

[95] J. Simon, **On Tape-Bounded Probabilistic Turing Machine Acceptors,**Theoretical Computer Science 16 (1981), pp. 75-91

[96] J. van de Snepscheut, **Deriving Circuits from Programs,** Third Caltech Conference on VLSI, Pasadena, CA, 1983

[97] R. I Soare, **Degrees and Structure of Speedable Sets,** Notices of the AMS, vol. 21 (1974), p. A-380. Abstract 74T-E40

[98] _____ **Computational Complexity, Speedable and Levelable Sets,** J. Symbolic Logic 42 (1977), pp. 545-563

[99] R. J. Solomonoff, **A Formal Theory of Inductive Inference, Part 1,** Information and Control, Vol. 7, No. 1, March 1964, pp. 1-22

[100] C. D. Thompson, **Area-Time Complexity for VLSI,** Proc. 11th Ann. ACM Symp. Theory of Computing, 1979

[101] J. E. Vuillemin, **A Combinatorial Limit to the Computing Power of VLSI Circuits,** Proceedings 21st Ann IEEE FOCS Symp., 1980, pp. 294-300

[102] W. Wilner, **Recursive Machines,** Xerox PARC SSL Internal Memorandum Jan 21, 1978

[103] A. C. Yao, **Some Complexity Questions Related to Distributive Computing,** Proc. 11th Annual ACM Symp. on Theory of Computing, 1979, pp. 209-213

[104] _____ **The Entropic Limitations on VLSI Computations,** Proc. 13th Annual ACM Symposium, Theory of Computing, 1981, pp. 308-311

[105] _____ **Theory and Applications of Trapdoor Functions,** Proc. 23rd IEEE Symp, Foundations of Computer Science, 1982, pp. 80-91

[106] B. H. Yin and J. W. Winchester, **The Establishmentand Use of Measures to Evaluate the Quality of Software Designs,** Proceedings ACM Software Quality Assurance Workshop, Vol 3, No. 5, 1978, pp. 45-52

[107] P. Young, **Optimization Among Provably Equivalent Programs,** Journal of the Association for Computing Machinery, Vol. 24, No. 4., pp. 693-700, October 1977

[108] A. K. Zvonkin and L. A. Levin, **Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms,** Russian Mathematical Surveys, Vol. 25, 1970