NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Report No. 32-999

Algorithmic Complexity

Richard Stanley

jpl
JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

September 1, 1966

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Report No. 32-999

Algorithmic Complexity

Richard Stanley

R Goldstein

R. M. Goldstein, *Manager*
Communications Systems Research Section

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

September 1, 1966

Prepared Under Contract No. NAS 7-100
National Aeronautics & Space Administration

CONTENTS

TABLES

FIGURE

ABSTRACT

The concept of algorithmic complexity that was introduced by Kolmogorov and expanded by Ofman provides a quantitative means of measuring the complexity of computing a discrete function—i.e., a function with finite domain and range. To be precise in the work reported here, it is assumed that the computation is done by a special type of finite-state machine, a (p, q) automaton. After reviewing the definitions in the field of algorithmic complexity, estimates are made for the maximum possible algorithmic complexity of a discrete function that can be computed on the simplest possible (p, q) automaton, a $(2, 2)$; this allows comparison of the algorithmic complexities relative to (p, q) automata and those relative to $(2, 2)$ automata. Next, bounds are obtained on the complexity of matrix multiplication. Finally, algorithmic complexity is related to the theory of permutation groups on the domain and range of a function, and various criteria are discussed for ensuring a function's having relatively low complexity.

I. INTRODUCTION

In this report, two fundamental problems of computer design are considered theoretically—minimizing the number of components (and, therefore, the cost) of the computer, and minimizing the computation time required. We define a mathematical object called a (p,q) automaton, where p and q are integers ≥ 2, which is to be regarded as an abstract model of a computer. The theory is easily modified to handle many other models of computers. Each (p,q) automaton computes a specific function and has a well defined number of components (stages) and computation time. Our object is to obtain upper and lower bounds on the number of stages and on the computation time required to calculate various functions. The least number of stages and least time required to compute a function f on any (p,q) automaton for fixed p and q is defined to be the algorithmic complexity of f relative to (p,q) automata. A precise definition of algorithmic complexity is given below.

In Section II, we consider the largest possible algorithmic complexity that a function can have; and in Section III, we discuss the complexity of matrix multiplication [over the field $GF(2)$]. Finally, in Section IV, by using the concept of equivalence of functions under permutation groups, we obtain criteria that guarantee that two functions have approximately the same complexity, and that a function has a relatively low complexity.

We begin with the necessary definitions. Let V_p^m denote the space of m-tuples over an alphabet of p symbols. Then, to define the algorithmic complexity of a function f: $V_p^k \rightarrow V_p^n$, we must first define a (p,q) automaton that computes f.

A. Definition of (p, q) Automaton

A (p,q) automaton, with $p,q \geq 2$, is an autonomous finite-state machine built of storage elements and gates. The storage elements, or *stages*, can be in one of p states at any time, corresponding to the p symbols of the alphabet. The gates determine the next state of the stages as a function of the immediately preceding states of, at most, q stages. In digital circuit terminology, there is, at most, one level of gating, and the gates have a

fan-in of, at most, q. The fan-out of each stage could be limited, also (Ref. 1), but this does not appear to add to the theory.

Further, the (p,q) automaton is defined as a (p-ary) shift register without feedback. That is, there is a numbering of the stages $Z_1, Z_2, \cdots, Z_n$ such that, if $j \geq i$, the state of each stage Z_i at time j equals the state at time i. The initial time (before the first shift) is $t = 1$. The stages that determine the state of a given stage at the next time-instant have no higher index than the stage determined; in fact, the restriction is made that the present state of a stage does not affect the next stage.

The machine is thought of as effecting a mapping $f: V_p^k \rightarrow V_p^n$ for some $k,n \geq 1$. Here, $k+n \leq N$; the first k stages correspond to V_p^k, and the last n to V_p^n. The initial state of the machine is as follows: an arbitrary element of V_p^k can be in the first k stages (the *input stages*) which, in practice, are usually loaded into an input register. The remaining $N-k$ stages are set, or cleared, to a fixed symbol, zero.

After a certain time—a time of, at most, N suffices—the states of all stages of the machine become, and remain, constant. At this time, the algorithm is declared computed for the given input k-tuples. The *output* n-tuple is then the contents of the last n output stages, often called the output register. The states of the remaining $N-n$ stages do not concern us at this time, but the machine must be cleared before a new computation starts.

This completes the (informal) definition of (p,q) automata. When $p=q=2$, the automaton becomes a binary feedbackless shift register built with, at most, one level of gates that are, at most, two-legged. We note that, although the class of finite-state machines defined above appears restrictive, nevertheless, any algorithm computable by any finite state machine in the most general form is also computable by some suitable (2,2) automaton without feedback (see Ref. 2).

B. Concept of Algorithmic Complexity

To define this concept, we will use Ofman's definition from Ref. 3.

Definition. Given integers $k, p, q > 1$, $n \geq 1$, and a function $f: V_p^k \rightarrow V_p^n$, consider all those (p, q) automata that realize f as a mapping from the first k stages at time 1 to the last n stages at some time when the state of the machine is stabilized and where the input and output stages are disjoint. Call this class of machines $M_f^{(p,q)}$, or M, for brevity. For B, an automaton in M, let N_B be the number of stages of B, and let T_B be the earliest time at which the output is available. Finally, let $A(f)$ be the set of all ordered pairs (N_B, T_B) for B in M, such that for no C in M is $N_C \leq N_B$, and such that $T_C \leq T_B$ with at least one inequality strict. Then, the set $A(f)$ is called the *algorithmic complexity of f*.

Remarks. It is clear that $A(f)$ is never empty. Ordinarily, $A(f)$ will have more than one element, although it is always a finite set. If C belongs to M, then there is a B in $A(f)$ that is at least as desirable as C for computing f—since some B in $A(f)$ computes f with, at most, the number of stages that C has in, at most, the time that C takes.

II. ALGORITHMICALLY COMPLEX MAPPINGS

Suppose that a class C of c functions $f: V_2^k \rightarrow V_2^n$ is given. Theorems 1 and 2 show that the algorithmic complexity of at least one of the functions in C cannot be too small. Theorem 1 is a generalization of a result in Ref. 4.

Theorem 1. *Let C be a class of c distinct functions $f: V_2^k \rightarrow V_2^n$ such that $c > 2^{16} \cdot 5^{10}$. Then, there is at least one function f_0 in C such that every (2,2) automaton realizing f_0 requires*

$$N > \frac{\log_2 c}{2 \log_2 \log_2 c}$$

stages.

Proof. A (2,2) automaton with N stages is determined by which pair of stages of the $\binom{N-1}{2}$ pairs each stage is affected, and by which one of the $2^{2^2} = 16$ functions $V_2^2 \rightarrow V_2^1$ specifies the relation. Hence, there are, at most, $\left[16\binom{N-1}{2}\right]^N$ (2,2) automata with N stages, so that if

$$\sum_{i=1}^{N} \left[8(i-1)(i-2)\right]^i < c \tag{1}$$

then some function $f_0 \in C$ requires greater than N stages to be computed.

Since each term of the left side of Eq. (1) is greater than twice the preceding term, we have

$$\sum_{i=1}^{N} \left[8(i-1)(i-2)\right]^i < 2 \cdot 8^N (N-1)^N (N-2)^N$$

$$< 2 \cdot 8^N N^{2N} \tag{2}$$

Now let N be the largest integer, such that

$$2 \cdot 8^N \cdot N^{2N} < c \tag{3}$$

It follows that

$$2 \cdot 8^{N+1} \cdot (N+1)^{2(N+1)} \geq c \tag{4}$$

Since $\log_2 x$ is a monotone increasing function, we may take the $\log_2 \log_2$ of each side of inequality Eq. (3):

$$\log_2(3N + 1 + 2N \log_2 N) < \log_2 \log_2 c \tag{5}$$

If $c \geq 17$, then both sides of inequality Eq. (5) are postive, and we have

$$\frac{1}{\log_2 N + \log_2\left(2 \log_2 N + 3 + \frac{1}{N}\right)} > \frac{1}{\log_2 \log_2 c} \tag{6}$$

Multiplication of inequality Eq. (6) by the $\log_2$ of inequality Eq. (4) gives

$$2N\left[\frac{\left(1 + \frac{1}{N}\right)\log_2(N+1) + \frac{3}{2} + \frac{2}{N}}{\log_2 N + \log_2\left(2 \log_2 N + 3 + \frac{1}{N}\right)}\right] > \frac{\log_2 c}{\log_2 \log_2 c} \tag{7}$$

An analysis of the bracketed quantity shows it to be < 1 when $N > 4$. Therefore,

$$N > \frac{\log_2 c}{2 \log_2 \log_2 c} \tag{8}$$

when $N > 4$ or $c > 2 \cdot 8^5 \cdot 5^{10}$, or approximately $2^{39.22}$. The above computations prove Theorem 1.

A similar result for the computaton time T is given in the following theorem.

Theorem 2. *Let C be a class of c distinct functions $f: V_2^k \rightarrow V_2^n$. Then, there is at least one function f_0 in C such that every (2, 2) automaton realizing f_0 requires a computation time*

$$T > \log_2 \log_2 c - \log_2 \log_2 k$$

Proof. It suffices to consider the case $n=1$, since all output bits may be computed separately and simultaneously.

If M is a (2,2) automaton that computes a function $f: V_2^k \rightarrow V_2^1$, then there is[1] another (2,2) automaton M' that computes the same function f with the same computation time as M, such that every stage of M' has, at most, one fan-out, and such that there are exactly 2^{T-t} stages of M' that are first affected at time t. It may be necessary, however, to repeat the same input bit into more than one input stage of M'. To construct M' from M, it is necessary only to label each stage of M, copy the one output stage, connect it to the two stages on which it depends, label these two stages to correspond to M, then repeat the

[1] A discussion of properties of automata of the type M appears in Ref. 1.

process until only stages with input labels remain unconnected to stages above them (see Fig. 1). At this point, we can add extraneous stages that merely double the multiplicity of each input stage until each input stage is removed by a time T from output.

Each of the $2^{T-1} - 1$ stages at time t, with $2 \leq t \leq T$, is determined by one of the 16 functions $g: V_2^2 \rightarrow V_2^1$. (The stages that repeat input correspond to one of these functions g; for example, $g(x_1, x_2) = x_1$.) The 2^{T-t} stages at time $t = 1$ are determined by which of the k input bits they admit. Hence, there are, at most,

$$16^{(2^{T-1}-1)}\, k^{2^{T-1}}$$

different functions $f: V_2^k \rightarrow V_2^1$ that can be realized in a time T on a (2,2) automaton.

Now, if

$$S(T) = \sum_{t=1}^{T} 16^{(2^{t-1}-1)}\, k^{2^{t-1}} < c$$

there is a function $f \in C$ that requires a time greater than T to be computed on any (2,2) automaton. But

$$S(T) < 2 \cdot 16^{2^{T-1}-1}\, k^{2^{T-1}}$$

so that

$$\log_2 S(T) < 2^{T+1}(1 + \tfrac{1}{4}\log_2 k)$$

and

$$\log_2 \log_2 S(T) < T + 1 + \log_2(1 + \tfrac{1}{4}\log_2 k)$$
$$\leq T + \log_2 \log_2 k \qquad (9)$$

Let T be the largest integer, such that

$$T < (\log_2 \log_2 c) - (\log_2 \log_2 k)$$

Then, from Eq. (9),

$$\log_2 \log_2 S(T) < \log_2 \log_2 c$$

Hence, there is a function requiring a time greater than T and, therefore, greater than or equal to

$$\log_2 \log_2 c - \log_2 \log_2 k$$

This completes the proof of Theorem 2.

Since there are $2^{n \cdot 2^k}$ distinct functions $f: V_2^k \rightarrow V_2^n$, there are immediate corollaries of Theorems 1 and 2.

Corollary 1. If $n \cdot 2^k > 39$, there is some function $f: V_2^k \rightarrow V_2^n$ such that every (2,2) automaton realizing f requires

$$N > \frac{n \cdot 2^k}{2 \log_2(n \cdot 2^k)}$$

stages.

Corollary 2. There is some function $f: V_2^k \rightarrow V_2^1$ such that every (2,2) automaton realizing f requires a computation time

$$T \geq k - \log_2 \log_2 k$$

We shall now obtain upper bounds on the algorithmic complexity of any function $f: V_2^k \rightarrow V_2^n$; combined with Corollaries 1 and 2, they give a narrow range for the maximum complexity possible for Boolean functions. The first theorem is essentially from G. N. Povarov (Ref. 5).

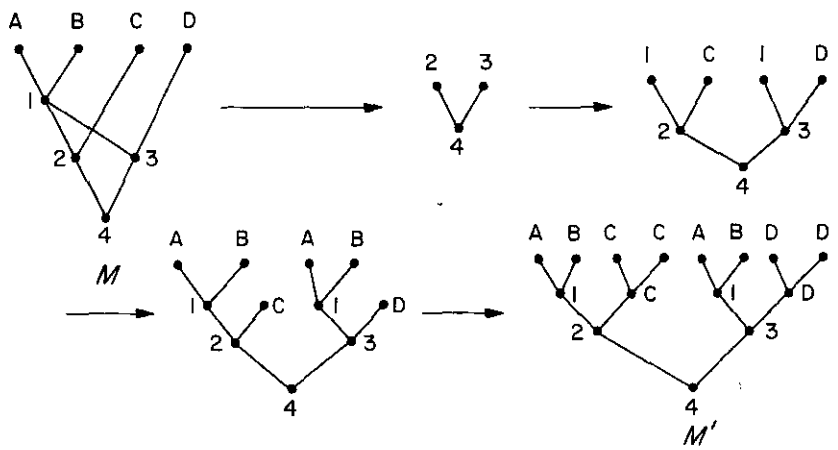

Fig. 1. Conversion of *M* into *M'*

Theorem 3. *Given $0 < \epsilon < 1$, for sufficient large k and $n < 2^{2^k - k}$, any function $f: V_2^k \to V_2^n$ can be realized on a (2, 2) automaton with*

$$N \leq \frac{4n\,2^k}{\log_2 n\,2^k}(1 + \epsilon)$$

Proof. G. N. Povarov proves in Ref. 5 that any function $f: V_2^k \to V_2^n$ can be realized by a (2, 2) automaton with

$$N = 2\left[n(2^{k-m} - 1) + 2^{2^m} - \sum_{i=0}^{m-1} 2^{2^i} + m - 2\right] \quad (10)$$

stages. Here, m is an arbitrary, positive integer less than k. Let m be the integral part and δ the fractional part of $\log_2 \log_2(n2^k) - \epsilon/5$. Then $m < k$ provided $n < 2^{2^k - k}$, and Eq. (10) becomes

$$N \leq 2\left[\frac{n \cdot 2^k\, 2^{\delta + \epsilon/5}}{\log_2 (n2^k)} + (n \cdot 2^k)^{2^{-\delta - \epsilon/5}}\right] \quad (11)$$

For sufficiently large k (depending only on ϵ), the second term is less than $\epsilon/5$ times the first. Since $2^\delta < 2$, and $2^{\epsilon/5} < 1 + 2\epsilon/5$ when $0 < \epsilon < 1$, Eq. (11) becomes

$$N \leq 2\left[\frac{n \cdot 2^k}{\log_2 (n \cdot 2^k)} 2\left(1 + \frac{2\epsilon}{5}\right)\left(1 + \frac{\epsilon}{5}\right)\right]$$

$$\leq \frac{4n \cdot 2^k}{\log_2 (n \cdot 2^k)}(1 + \epsilon)$$

and Theorem 3 follows.

Corollary 1 and Theorem 3, together, determine the maximum number of stages required to compute a function within a factor of 8.

Theorem 4. *Any function $f: V_2^k \to V_2^n$ can be realized on a (2, 2) automaton with computation time*

$$T < k + \log_2 k + 1$$

Proof. It suffices, as usual, to consider the case $n = 1$. Let S_0 and S_1 denote the set of k-tuples in V_2^k whose image under f is 0 and 1, respectively. One of S_i satisfies $|S_i| \leq 2^{k-1}$, say S_0. For each given $x \in S_0$ a time $T < \log_2 k + 2$ is required to determine if the input k-tuple is equal to x. In the automaton performing this computation, every stage has only one fan-out and every stage (except input) is an AND gate. The output bit is a 1 if $x \in S_0$; it is a 0 if $x \notin S_0$. The $|S_0|$ output bits, 0 or 1, can all be computed simultaneously. To determine if one of these $|S_0|$ bits is a 1 requires an additional time $T \leq k - 1$, since $|S_0| \leq 2^{k-1}$. Here again, every stage has one fan-out, only; but now, every stage is an (inclusive) OR gate. The final output bit is 0 if $x \in S_0$, and 1 if $x \notin S_0$, i.e., the function value of x. Thus, the entire computation time is

$$T < k + \log_2 k + 1$$

which completes the proof.

The above theorems allow us to compare the performance of (2, 2) automata and (p, q) automata. In order to compute a function $f: V_p^k \to V_p^n$ on a (2, 2) automaton, it is necessary to encode the p symbol alphabet onto some subset of r-tuples from V_2^r, where $2^{r-1} < p \leq 2^r$. Each stage of the original (p, q) automaton computing f is replaced by r binary stages. The function f now has domain V_2^{kr} and range V_2^{nr}. Let $N(p, q)$ and $T(p, q)$ denote the number of stages and computation time of a given (p, q) automaton as defined in the Introduction. It is stated in Ref. 3 that there exist constants C_1 and C_2 depending only on p and q, such that

$$N(2, 2) \leq C_1\, N(p, q)$$

$$T(2, 2) \leq C_2\, T(p, q)$$

It is now easy to obtain bounds for C_1 and C_2.

Theorem 5. *For any $0 < \epsilon < 1$ and p^q sufficiently large, we have*

$$N(2, 2) \leq (1 + \epsilon)\frac{8p^q}{q} N(p, q)$$

Proof. We first replace each p-ary stage by $r \leq \log_2 p + 1$ binary stages. Each binary stage now depends on qr stages, so that our (p, q) automaton with $N(p, q)$ stages has become a $(2, qr)$ automaton with $rN(p, q)$ stages. We replace each stage, together with the qr stages on which it depends, by a (2, 2) automaton with qr input stages and 1 output stage. By Theorem 3, this requires at most $[(4 \cdot 2^{qr})/qr](1 + \epsilon) - qr - 1$ additional stages for each stage replaced, when $qr \leq \log_2 p^q + q$ is sufficiently large. Hence, our (p, q) automaton has become a (2, 2) automaton with

$$N(2, 2) \leq \left[\frac{4 \cdot 2^{qr}}{qr}(1 + \epsilon) - qr - 1\right] r\, N(p, q) + rN(p, q)$$

stages. Since $r \leq \log_2 p + 1$, we get

$$N(2, 2) \leq (1 + \epsilon)\frac{8p^q}{q} N(p, q)$$

and, thus, complete the proof.

Theorem 6.

$$T(2,2) \leq (q(1 + \log_2 p) + \log_2 q(1 + \log_2 p))\, T(p, q)$$

Proof. In the proof of Theorem 5, the conversion of the (p, q) automaton to a $(2, qr)$ automaton does not increase the computation time. If we replace each stage, as well as the qr stages on which it depends, by a (2, 2) automaton as constructed in Theorem 4, then by Theorem 4, for each unit of time in the $(2, qr)$ automaton, we are adding an additional time less than $qr + \log_2 qr - 1$ (2 is subtracted from the upper bound of Theorem 4 because the input and output are part of the $(2, qr)$ automaton). Therefore,

$$T(2,2) \leq (qr + \log_2 qr - 1)\, T(p,q) + T(p,q)$$

$$\leq [q(\log_2 p + 1) + \log_2 q\,(\log_2 p + 1)]\, T(p,q)$$

completing the proof.

III. ALGORITHMIC COMPLEXITY OF LINEAR MAPPINGS

Let **A** be here a fixed $r \times s$ matrix over the field $GF(2)$. Consider **A** as a mapping **A**: $V_2^{st} \to V_2^{rt}$ performing the operation of matrix multiplication whose domain consists of all $r \times t$ matrixes over $GF(2)$ and whose range consists of $r \times t$ matrixes over $GF(2)$. Since there are 2^{rs} distinct $r \times s$ matrixes **A**, the following theorem is an immediate consequence of Theorem 1.

Theorem 7. *If $rs > 39$, then there is some $r \times s$ matrix **A** over GF(2), such that every (2, 2) automaton performing the matrix multiplication **A**: $V_2^{st} \to V_2^{rt}$ requires at least $rs/(2 \log_2 rs)$ stages.*

By modifying Theorem 3, L. R. Welch (Ref. 6) has obtained a converse to Theorem 7, of which the following theorem is a slight modification.

Theorem 8. *Any matrix multiplication **A**: $V_2^{st} \to V_2^{rt}$ can be realized on a (2, 2) automaton with*

$$N \leq rt\left(\frac{2s}{[\log_2 r]} + 1\right)$$

stages, where brackets denote the integer part.

Proof. It suffices to prove the theorem for $t = 1$, since **A** can produce each of the t columns of the range independently, to give a factor of t. We claim that there is a (2, 2) automaton M, with m input stages and 2^m stages in all, such that the stages realize all 2^m linear functionals of the input variables into $GF(2)$. This assertion is clearly true for $m = 1$; a stage with no inputs produces the zero functional, whereas the input stage itself produces the remaining linear functional of one variable. Suppose the hypothesis is true for $m - 1$. Then, there is an automaton M_0 with 2^{m-1} stages realizing all linear functionals of $m - 1$ variables $x_1, \cdots, x_{m-1}$.

Adjoin $2^{m-1} - 1$ stages, each of which depends on x_m and one stage of M_0, excluding the stage which realizes the zero functional. Each new stage computes the operation of mod 2 addition. Clearly, the enlarged automaton has 2^m stages that realize all linear functionals of m variables $x_1, \cdots, x_m$. The proof of the assertion now follows by induction.

First, for any given s_0, $1 \leq s_0 \leq s$, group the s variables into sets of size s_0 with the last group of size $\sigma = s - s_0\,[s/s_0]$ (brackets denote the integer part). Then, for each set, form all linear functionals using

$$\frac{s - \sigma}{s_0}\, 2^{s_0} + 2^{\sigma}$$

stages. Now, each of the r output functions is a linear combination of $[(s - \sigma)/s_0] + 1$ functions. This combination can be constructed with $(s - \sigma)/s_0$ stages. The total number of stages is, then,

$$N = r\,\frac{s - \sigma}{s_0} + \frac{s - \sigma}{s_0}\, 2^{s_0} + 2^{\sigma} \qquad (12)$$

Let s_0 equal the integer part of $\log_2 r$, and let δ equal the fractional part of $\log_2 r$. Equation (12) becomes the inequality

$$N \leq \frac{rs}{[\log_2 r]}\,(1 + 2^{-\delta}) + r$$

$$\leq \frac{2rs}{[\log_2 r]} + r$$

and Theorem 8 follows.

In particular, if $r \leq 2^{s+1}$, then

$$N \leq \frac{3rst}{\log_2 r - 1}$$

If $r > 2^s$, then at least $r - 2^s$ rows of **A** must be duplicated, which requires only t stages each, once the rest of the multiplication has been computed. Hence, when $r > 2^s$, we get

$$N \leq t\left(\frac{2 \cdot 2^s \cdot s}{\log_2 2^s} + 2^s\right) + t(r - 2^s)$$
$$= t\,(2^{s+1} + r)$$

The ratio of the upper and lower bounds obtained in Theorems 7 and 8 grows arbitrarily large, even for fixed t. For small r and large s, the lower bound becomes unrealistic; e.g., when $s > 2^{\alpha r}$, the lower bound is less than $s/2\alpha$, while $2s - 1$ stages are needed when

$$\mathbf{A} = \begin{Bmatrix} 1 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{Bmatrix}$$

In the case in which $r = s$ and $t = 1$, i.e., when **A** is a square matrix that operates on a vector, the two bounds differ by a factor of 8 as $r \to \infty$:

$$\frac{r^2}{4 \log_2 r} \leq N \leq \frac{2r^2}{[\log_2 r]} + r$$

The upper bound also becomes unrealistic for small n and large k. The next theorem gives another upper bound, which is more precise in this case. A similar result is stated by L. R. Welch (Ref. 6) without proof; Welch does not include the input stages in his bound.

Theorem 9. *Any matrix multiplication* $A: V_2^{st} \to V_2^{rt}$ *can be realized on a (2, 2) automaton with*

$$N \leq \left(\frac{r2^r}{[\log_2 r]} + r - 2^r + 2s - 1\right) t$$

stages.

Proof. As before, we may assume $t = 1$. Let $\mathbf{A}_{i_1 i_2 \cdots i_k}$ denote the collection of all those columns $c_{j_1}, c_{j_2}, \cdots, c_{j_u}$ of **A** which have a 1 in their $i_1, i_2, \cdots, i_k$ coordinates and a 0 elsewhere. For each such collection $c_{j_1}, \cdots, c_{j_u}$, construct an automaton with less than $u - 1$ stages (excluding input) which adds mod 2 the u input variables $x_{j_1}, \cdots, x_{j_u}$. We now have 2^{r-1} subautomata with a total of, at most, $2s - 1$ stages, since they all share s input stages, and any $\mathbf{A}_{i_1 i_2 \cdots i_k}$ is disjoint from any other one.

It remains only to connect various subautomata by mod 2 addition to compute the r output bits. It is not hard to see that these connections are equivalent to multiplication by a special matrix with r rows and 2^{r-1} columns. If two columns of this special matrix are identical, then they both contain only 0's. By Theorem 8, it requires less than $r2^r/[\log_2 r] + r$ stages to perform this multiplication, of which 2^r have already been used for input. Hence, we have used, altogether,

$$N \leq \frac{r2^r}{[\log_2 r]} + r - 2^r + 2s - 1$$

stages. This computation completes the proof of Theorem 9.

The problem of finding the least computation time for matrix multiplication is not as interesting as the problem of finding the number of stages, since each entry of the output matrix can be computed simultaneously. Thus, the least time T is simply the least time required to compute the scalar product $z \cdot x$, where z is a fixed s-tuple and x is an input s-tuple. If z contains exactly α ones, then the least possible computation time is clearly $\{\log_2 \alpha\} + 1$, where $\{y\}$ denotes the rounding upward function, i.e., the least integer greater than, or equal to, y. We state the above observations as a theorem.

Theorem 10. *If* A *is an* $r \times s$ *matrix over GF(2), then the matrix multiplication* $A: V_2^{st} \to V_2^{rt}$ *can be realized by a (2, 2) automaton with a computation time*

$$T = 1 + \{\log \alpha\} \tag{13}$$

where α *is the greatest number of ones appearing in any row of* A*, and* $\{x\}$ *denotes the rounding upward function. Furthermore, every (2, 2) automaton realizing* A *requires a computation time at least as large as that given in Eq. (13). In particular, some* $r \times s$ *matrixes require a time* $T = 1 + \{\log s\}$*, but no* $r \times s$ *matrix requires more time.*

IV. CRITERIA FOR SMALL COMPLEXITY

There are a number of properties that a Boolean function $f: V_2^k \to V_2^n$ may have which guarantee that it have a relatively low algorithmic complexity. A knowledge of these properties also narrows the search for specific functions of high complexity. We shall restrict our concern, generally, to the case $n = 1$; when $n > 1$, the automaton may be regarded as n independent automata sharing the same input.

We begin with the concept of *equivalence of functions under permutation groups on their domain and range.* Let G be a permutation group on the set of all k-tuples $(x_1, \cdots, x_k) \in V_2^k$, and let H be a permutation group on the set of all n-tuples $(y_1, \cdots, y_n) \in V_2^n$. We say that the functions $f: V_2^k \to V_2^n$ and $g: V_2^k \to V_2^n$ are *equivalent with G on the domain and H on the range* if for some fixed $\pi \in G$, $\sigma \in H$ we have

$$\sigma f[\pi(x)] = g(x) \tag{14}$$

for all $x \in V_2^k$. We use the notation (G, H) to denote a group G acting on the domain and a group H acting on the range of a Boolean function; if Eq. (14) holds, then, we also say that f and g are *equivalent under (G, H).* (It is easily seen that the relation that we have called equivalence under (G, H) is, indeed, an equivalence relation.) Let $E(G, H)$ denote the number of equivalence classes induced by the equivalence relation (G, H).

We shall be concerned with the following permutation groups G, H:

- $G = C_2^k$ the group of all 2^k complementations of input variables: This group is isomorphic to the elementary abelian group of order 2^k.
- $G = S_k$ the symmetric group on the k input variables: This is the group of all permutations of input bits.
- $G = G_k$ the smallest group containing both C_2^k and S_k: G_k is the semi-direct product of C_2^k and S_k (see Theorem 4 in Section 6 of Ref. 7).
- $G = GL_k$ the full linear group on the input variables: This is the group of all nonsingular linear transformations of the input variables.
- $G = A_k$ the affine group on the input variables: This is the smallest group containing both C_2^k and GL_k.
- $H = 1$ the group consisting only of the identity: In other words, two elements y_1, y_2 of the range are considered equivalent if, and only if, $y_1 = y_2$.
- For $n = 1$, $H = N$ the group of complementations on the range: N is isomorphic to the cyclic group of order 2.

For some applications of these groups to the theory of Boolean functions, the reader is referred to Harrison (Refs. 7–10) and to Hertzig and Dean[2]. An immediate application to algorithmic complexity is given by the following theorem.

Theorem 11. *The algorithmic complexity M_f of $f: V_2^k \to V_2^1$ is invariant under (G_k, N); i.e., if f and g are equivalent under (G_k, N), then f and g have exactly the same algorithmic complexity on a (2, 2) automaton.*

Proof. A permutation $\pi \in S_k$ of the variables merely corresponds to a relabeling of the stages of the automaton and, consequently, does not affect the algorithmic complexity. If a complementation $\sigma \in C_2^k$ is carried out on certain variables, then we need only appropriately alter the functions $f_0: V_2^2 \to V_2^1$ of pairs of input variables that correspond to the first step of the computation. Again, the algorithmic complexity is not affected. The complementation on the range may be incorporated into the last step of the computation when necessary, without altering the complexity. Since every permutation in G_k is the product of permutations in C_2^k and S_k, the proof of Theorem 11 is complete.

The group GL_k (and hence, A_k) does not preserve algorithmic complexity. For instance, when $k > 1$, the permutation taking $x_1 \to x_1 + x_2 + \cdots + x_k$ results in an increase of complexity in computing $f(x_1, \cdots, x_k) = x_1$. The next theorem, however, shows that functions equivalent under (A_k, N) cannot differ in their complexity by too great an amount.

Theorem 12. *If $f: V_2^k \to V_2^1$ and $g: V_2^k \to V_2^n$ are equivalent under (A_k, N), and if f can be computed with N_f stages in a time T_f on a (2, 2) automaton, then g can be*

[2]Hertzig, D., and Dean, R., "Decompositions and Equivalences of Numerical Functions on a Vector Space Over GF(2)," unpublished paper (private communication from Professor Dean, California Institute of Technology).

computed with N_g stages in a time T_g on a (2, 2) automaton, where

$$N_g \leq N_f + \frac{2k^2}{[\log_2 k]}$$

$$T_g \leq T_f + \{\log_2 k\}$$

Proof. By assumption, there is a fixed (nonsingular) matrix **A**, a fixed vector **v**, and a fixed permutation $\pi \in N$, such that

$$g(x) = \pi f(\mathbf{Ax} + \mathbf{v})$$

for all **x** in V_2^k. Hence, g can be computed by first computing **Ax** + **v**, then computing $\pi f(\mathbf{Ax} + \mathbf{v})$. The permutation π can be incorporated into the last step of the computation of *f*, so $N_{\pi f} = N_f$ and $T_{\pi f} = T_f$. Let N_A and T_A denote the number of stages and computation time for computing **Ax** + **v**. Since the output of **Ax** + **v** corresponds to the input of *f*, we have

$$N_g \leq N_f + N_A - k$$

$$T_g \leq T_f + T_A - 1 \qquad (15)$$

The addition of *v* to **Ax** can be incorporated into the last step of the computation of **Ax**, so that we can take for N_A and T_A the bounds for matrix multiplication obtained in Theorems 8 and 9. Substitution of these results into Eq. (15) gives Theorem 12.

Theorems 11 and 12 result in a substantial decrease in the number of functions $f: V_2^k \rightarrow V_2^1$ that have to be considered when dealing with algorithmic complexity. Rather than considering 2^{2^k} functions, we may consider, instead, $E(G_k, N)$ or $E(A_k, N)$ classes of functions. Table 1 reproduces the values of $E(G_k, N)$ and $E(A_k, N)$ obtained by Harrison (Ref. 8) for $1 \leq k \leq 6$ and compares these with those of 2^{2^k}. A method of calculating these values, based on a theorem of De Bruijn (Ref. 11) is also discussed in Ref. 9.

To discuss two properties of functions $f: V_2^k \rightarrow V_2^1$ which tend to make their complexities small, we begin with two definitions.

Definition: Length of Function. *Let $f: V_2^k \rightarrow V_2^1$.* We define the *length* $|f|$ of *f* to be the number of terms appearing when *f* is written out as a mod 2 sum of products. For instance,

$$|0| = 0$$

$$|1| = 1$$

$$|x_1 x_2 + x_3| = 2$$

$$|x_1 x_2 + x_1 + x_2 + 1| = 4$$

Furthermore, we define the length of *f* under (*G*, *H*), denoted by $|f|_{(G,H)}$, to be the minimum length of all functions in the equivalence to which *f* belongs under (*G*, *H*).

Definition: Measures μ and $|\mu|$ of f. *Let $f: V_2^k \rightarrow V_2^1$, and* let S_0 and S_1 denote the set of elements in V_2^k, which are mapped into 0 and 1, respectively. (Clearly, $S_0 \cup S_1 = V_2^k$ and $S_0 \cap S_1 = \phi$.) If s_i denotes the number of elements in S_i, then we define two measures, μ and $|\mu|$, of *f* as follows:

$$\mu(f) = s_1 - s_0$$

$$|\mu|(f) = |s_1 - s_0|$$

The measure μ was defined and discussed by Hertzig and Dean[2], but for our purposes $|\mu|$ will be more convenient.

A function *f* is called *neutral* (Hertzig and Dean use the term *balanced*) if $\mu(f) = |\mu|(f) = 0$. Some properties

Table 1. Number of classes of Boolean functions

No. of variables *k*	Value of 2^{2^k}	Size of classes of functions[a]	
		$E(G_k, N)$	$E(A_k, N)$
1	4	2	2
2	16	4	3
3	256	14	6
4	65,536	222	18
5	4,294,967,296(4.3 × 10^9)	616,126(6.2 × 10^5)	206
6	18,446,744,073,709,551,616(1.8 × 10^{19})	200,253,952,527,184(2.0 × 10^{14})	7,888,299(7.9 × 10^6)

[a] From Harrison (Ref. 8).

of neutral functions are given in Ref. 8 and the paper cited in Footnote 2. For instance, it is shown by Harrison in Ref. 8 (as a special case of his Lemma 11) that the number of neutral classes of functions $f: V_2^k \to V_2^1$ under $(G_k,1)$ is greater than, or equal to,

$$\frac{1}{2^k\,k!}\binom{2^k}{2^{k-1}} \sim \sqrt{\frac{2}{\pi}}\,\frac{2^{2^k-(3k/2)}}{k!}$$

It is easy to see that the measure $|\mu|(f)$ is invariant under (G_k, N); the length $|f|$, however, is not invariant.

The relationship of the length of a function to its algorithmic complexity is given by the following theorem.

Theorem 13. *Let $f: V_2^k \to V_2^1$ and set $\ell = |f|_{(G_k, N)}$. Then, there is a (2,2) automaton realizing f with N_f stages in a time T_f, where*

$$N_f \leq k(\ell + 1) - 1$$

$$T_f \leq 1 + \{\log_2 k\} + \{\log_2 \ell\}$$

Proof. Each of the ℓ terms is a product of, at most, k variables. Each term therefore requires $N \leq 2k - 1$, $T \leq 1 + \{\log_2 k\}$. Since input need not be repeated, all ℓ terms can be computed with $N \leq (2k-1)\,\ell - k(\ell - 1) = k(\ell + 1) - \ell$, $T \leq 1 + \{\log_2 k\}$. To add the ℓ terms requires an additional $N \leq \ell - 1$, $T \leq \{\log_2 \ell\}$. Hence,

$$N_f \leq k(\ell + 1) - \ell + (\ell - 1) = k(\ell + 1) - 1$$

$$T_f \leq 1 + \{\log_2 k\} + \{\log_2 \ell\}$$

as required.

By using a slight modification of the proof of Theorem 8 —considering mod 2 multiplication rather than mod 2 addition—it can, in fact, be shown that

$$N_f \leq \frac{2k\ell}{[\log_2 k]} + k + \ell - 1$$

Loosely speaking, Theorem 13 states that short functions (functions with small ℓ) have low complexities. The question of what, in fact, is the maximum value $L = L(k)$ of $|f|_{(G_k,N)}$, as f ranges over V_2^k is of interest. It seems plausible to conjecture the asymptotic formula $L(k) \sim 2^{k-1}$. In fact, it may even seem reasonable to conjecture that $L(k) = 2^{k-1}$; this equality is valid for $1 \leq k \leq 3$. A counterexample to this latter conjecture, however, is provided by

$$f(x_1,x_2,x_3,x_4) = x_1x_2x_3x_4 + x_1x_2x_3 + x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4 + x_4$$

for here, $|f|_{(G_4,N)} = 9$. We can, however, obtain fairly precise bounds for $L(k)$. We first need an inequality of interest in its own right.

Lemma 1. If $x_0 > 0$, then $(1 + x_0)^{2(1+x_0)} < (1 + 2x_0)^{1+2x_0}$.

Proof. Define α by $(1 + x_0)^{1+x_0} = e^{\alpha x_0}$, and let $g(x) = (1 + x)\log(1 + x) - \alpha x$. Note that $g(x_0) = 0$. Now $g'(x) = \log(1 + x) + 1 - \alpha$, so $g'(x_0) = (\alpha x_0)/(1 + x_0) + 1 - \alpha = 1 - \alpha/(1 + x_0)$. If $1 + x_0 < \alpha$, then $(1 + x_0)\log(1 + x_0) < \alpha \log(1 + x_0) < \alpha x_0$, since $\log(1 + x_0) < x_0$ for $x_0 > 0$. This contradicts $(1 + x_0)\log(1 + x_0) = \alpha x_0$; hence, $1 + x_0 \geq \alpha$ and $g'(x_0) \geq 0$. Since $g'(x)$ is increasing for $x \geq 0$, we have $g(x) \geq 0$ for $x \geq x_0$. If $x = 2x_0$,

$$(1 + 2x_0)^{1+2x_0} \geq [\exp(\alpha x_0)]^2 = (1 + x_0)^{2(1+x_0)}$$

which, thus, proves the lemma.

Theorem 14. *Given $\epsilon > 0$, then for sufficiently large k we have*

$$\frac{2^{k-1}}{1+\epsilon} < L(k) < \frac{4}{3}\cdot 2^{k-1}$$

(The upper bound is valid for all k.)

Proof. We first establish the lower bound. Since G_k has order $2^k\,k!$ and N has order 2, there are, at most, $2^{k+1}\,k!$ functions in equivalence class under (G_k, N). There are $\binom{2^k}{i}$ functions of length i, so that if

$$\sum_{i=1}^{\ell}\binom{2^k}{i} < \frac{2^{2^k}}{k!\,2^{k+1}} \tag{16}$$

then $\ell < L(k)$. Put $\ell = \frac{2^{k-1}}{1+\epsilon}$ and let $k \to \infty$ in the left side of Eq. (16). Although it is possible to obtain an exact asymptotic formula for the left side of Eq. (16), for our purposes, the following estimate will suffice:

$$\sum_{i=1}^{\ell}\binom{2^k}{i} < \ell\binom{2^k}{\ell} \sim \left(\frac{2^k}{3\pi}\right)^{1/2}\left[\frac{2(1+\epsilon)}{(1+2\epsilon)^{1-\frac{1}{2(1+\epsilon)}}}\right]^{2^k} \tag{17}$$

by Stirling's formula. By Lemma 1 the expression in brackets is less than 2, so that for large k, the right-hand side of Eq. (17) is less than $2^{2^k}/k!\,2^{k+1}$. This establishes the lower bound.

The upper bound is obtained by induction on k. It clearly holds for $k = 0$, as $L(0) = 0$. Assume the result

valid for $k-1$. Let $f: V_2^k \to V_2^1$ be a function that achieves $L(k) = |f|_{(G_k,N)}$, and write

$$f(x_1,\cdots,x_k) = x_1 f_1(x_2,\cdots,x_k) + f_2(x_2,\cdots,x_k).$$

Transform the input, if necessary, by a permutation in G_k so that $|f_1| \leq L(k-1) < \frac{4}{3}\cdot 2^{k-2}$. Hence,

$$L(k) < \frac{4}{3}\cdot 2^{k-2} + |f_2| \tag{18}$$

Clearly, f is equivalent under (G_k, N) to $x_1 f_1 + f_1 + f_2$, for this results from the complementation $x_1 \to x_1 + 1$. It is not hard to see that $|f_1 + f_2| \leq 2^k - |f_1| - |f_2|$. Hence, we also have

$$\begin{aligned} L(k) &< |f_1| + (2^k - |f_1| - |f_2|) \\ &= 2^k - |f_2| \end{aligned} \tag{19}$$

Adding inequality Eqs. (18) and (19) gives $2L(k) < (4/3)\cdot 2^{k-2} + 2^k = 4/3\cdot 2^k$. Therefore, $L(k) < 4/3\cdot 2^{k-1}$, and the proof follows by induction. Theorem 14 is so proved.

We now show the relationship between the measure $|\mu|$ and algorithmic complexity.

Theorem 15. *Let $f: V_2^k \to V_2^1$. If $|\mu|(f) = m$; then, there exists a (2,2) automaton realizing f with N_f stages and a computation time T_f, where*

$$N_f \leq \frac{1}{2}k(2^k - m) + k - 1$$

$$T_f \leq \{\log_2 k\} + \{\log_2(2^k - m)\}$$

Proof. If $|\mu|(f) = m$, then one of S_0 or S_1, say S_0, has $\frac{1}{2}(2^k - m)$ elements and the other one has $\frac{1}{2}(2^k + m)$ elements. We now use the procedure of the proof of Theorem 4 to compute f. To determine whether the input is equal to a given fixed element of S_0 requires

$$N \leq 2k - 1$$

$$T \leq 1 + \{\log_2 k\}$$

Hence, to test all $(2^k - m)/2$ elements of S_0, without repeating input, requires

$$N \leq k + \frac{1}{2}(2^k - m)(k-1)$$

$$T \leq 1 + \{\log_2 k\} \tag{20}$$

At this point we have $\frac{1}{2}(2^k - m)$ stages, of which exactly one is equal to 1 if, and only if, the input was in S_0; otherwise, all $(2^k - m)/2$ stages are 0. To determine whether

Table 2. Boolean functions of $k \leq 3$ variables under (G_k, N)

a. One variable

$k=1$	Representative function or minimal length	No. in class	Measure $\|\mu\|$
1	0	2	2
2	a	2	0
		4	

b. Two variables

$k=2$	Representative function or minimal length	No. in class	Measure $\|\mu\|$
1	0	2	4
2	a	4	0
3	$a+b$	2	0
4	ab	8	2
		16	

Table 2. (Cont'd)

c. Three variables

$k=3$	Representative function or minimal length	No. in class	Measure $\|\mu\|$
1	0	2	8
2	a	6	0
3	$a+b$	6	0
4	ab	24	4
5	$a+b+c$	2	0
6	$ab+c$	24	0
7	$ab+ac$	24	0
8	$ab+bc+a$	24	0
9	$ab+ac+bc$	8	0
10	$ab+ac+bc+a$	8	4
11	abc	16	6
12	$abc+a$	48	2
13	$abc+a+b$	48	2
14	$abc+a+b+c$	16	2
		256	

one of these $(2^k - m)/2$ stages is 1 requires an additional N,T as follows:

$$N \leq \frac{1}{2}(2^k - m) - 1$$

$$T \leq \left\{\log_2 \frac{2^k - m}{2}\right\} = \{\log_2(2^k - m)\} - 1 \tag{21}$$

Combination of inequality Eqs. (20) and (21) gives the statement of the theorem, completing the proof.

Theorem 15, in effect, states that highly non-neutral functions have low complexities, which is not surprising. Theorems 13 and 15 combined suggest that the place to look for algorithmically complex functions is among the nearly neutral functions of long length.

Table 2 describes the equivalence classes of the Boolean functions of k variables under (G_k, N) for $k \leq 3$. Included are representative functions of minimal length in each equivalence class, the number of functions in each class, and the common measure $|\mu|$ of the functions in each class.

V. SUMMARY

We have defined the concept of algorithmic complexity in order to estimate the size and the time required for a computer to calculate various functions. In general, we would like to determine the exact complexity of any given function, but this seems much too difficult. In this report, therefore, we have discussed methods of approximating the complexity of special classes of functions.

Using a (2, 2) automaton as a model for a computer, we have determined the maximum size (number of stages) required to compute any function within a factor of 8; at the same time, we have determined the maximum time required asymptotically. An important special class of functions is made up of linear mappings; we have obtained bounds on their algorithmic complexity which, in the case of multiplication by a square matrix, are accurate within a factor of 8. Many functions have complexities considerably less than the upper bounds obtained in Section II; therefore, recognition of such functions is important. We have shown that the complexities of functions equivalent under certain permutation groups do not differ significantly.

In many cases, a function may be seen to have low complexity when it is compared with an equivalent function whose complexity has been estimated by some other means. Measure and length are two such criteria used to guarantee that a function has low complexity.

REFERENCES

1. Stanley, R., "The Notion of a (p, q, r) Automaton," *Supporting Research and Advanced Development, SPS No. 37-35,* Vol. IV, Jet Propulsion Laboratory, Pasadena, Calif., October 31, 1965, pp. 292–298.

2. Golomb, S. W., *The Shift Register as a Finite-State Machine,* USCEE No. 122, University of Southern California, Los Angeles, Calif., 1965.

3. Ofman, Yu, "On the Algorithmic Complexity of Discrete Functions," *Sov. Phys. Daklady, Cyb. Cont. Theory 7* (1963), pp. 589–591 (Translated).

4. Stanley, R., "New Results of Algorithmic Complexity," *Supporting Research and Advanced Development, SPS No. 37-34,* Vol. IV, Jet Propulsion Laboratory, Pasadena, Calif., August 31, 1965, pp. 298–305.

5. Povarov, G. N., "Synthesis of Contact Multipoles," paper reviewed in *IRE Trans. Circuit Theory, 3* (1956), p. 78.

6. Welch, L. R., "Asymptotic Algorithmic Complexity," *Supporting Research and Advanced Development, SPS No. 37-39,* Vol. IV, Jet Propulsion Laboratory, Pasadena, Calif., June 30, 1966.

7. Harrison, M. A., "The Number of Transistivity Sets of Boolean Functions," *Soc. Ind. Appl. Math, J., II,* No. 3 (September 1963), pp. 806–828.

8. Harrison, M. A., On the Classification of Boolean Functions by the General Linear and Affine Groups, *Technical Note,* College of Engineering, University of Michigan, Ann Arbor, Mich., September 1962.

9. Harrison, M. A., "The Number of Equivalence Classes of Boolean Functions Under Groups Containing Negation," *IEEE Trans. Electron. Computers, EC-12,* No. 5 (October 1963).

10. Stanley, R., "Further Results on the Algorithmic Complexity of (p, q) Automata," *Supporting Research and Advanced Development, SPS No. 37-35,* Vol. IV, Jet Propulsion Laboratory, Pasadena, Calif., October 31, 1965, pp. 298–304.

11. De Bruijn, N. G., "Generalization of Polya's Fundamental Theorem in Enumerative Combinatorial Analysis," *Koninklijke Nederlandse Akademe van Wetenschappen 62* (1959), pp. 59–69.