Contributions to a Mathematical Theory of Complexity

L. H. Harper
University of California at Riverside

J. E. Savage
Brown University

This article is another in a series that attempts to define precisely and investigate the "computational complexity" of a general class of problems which includes many problems that occur in the DSN control center. Specific DSN control center questions that a theory of computational complexity will help define and answer include the problem of the optimum mix of core, disk, and drum storage in the control center, and the intelligent allocation of computational resources to flight projects of differing complexity in such a way that simultaneous real-time computing commitments can be made. This article shows that there exists such a theory which is capable of providing important information about the true complexity of several classes of non-trivial problems.

I. Introduction

The Deep Space Network depends critically upon computation: computation of trajectories, decoding of telemetry, processing of ranging data, etc. For every computation we make, however, we should also ask, "Could this be done more efficiently?" Sometimes we can answer affirmatively by producing a better algorithm; rarely can we assert that no improvement is possible. It is the object of this article to present the elements of a theory of complexity which may lead to a satisfactory answer to this question in many cases. In a previous article (Ref. 1), we began the study of complexity; this article continues the work.

Specific DSN control center questions which a theory of computational complexity will help define and answer include the problem of the optimum mix of core, disk, and drum storage in the control center, and the intelligent allocation of computational resources to flight projects of differing complexity, in such a way that simultaneous real-time computing commitments can be made. This article shows that there exists such a theory which is capable of providing important information about the true complexity of several classes of non-trivial problems.

In *Section II*, we define the *computational complexity* and the *computation time* of any Boolean function of N variables, and contrast a theorem of Lupanov which says that "most" such functions have complexity near 2^N, with a simple lemma which shows that any function which really depends on N variables has complexity at least $\rho(N-1)$ for a certain constant ρ. In *Section III* we present a condition which guarantees that a function is rather complex, sometimes as complex as $N^2/\log N$.

Finally, in *Section IV*, we apply the results of the previous sections to two famous search problems of combinatorial mathematics: the "marriage problem" and the "traveling salesman's problem." We establish a lower bound of $0(n^3)$ on the combinational complexity (with logic fan-out one) for the marriage problem. The classical "alternating paths" technique gives an upper bound of $0(n^4)$ for the combinational complexity (with unlimited fan-out) for this problem. Hopefully, more work will eliminate the fan-out restrictions and close the gaps between upper and lower bounds.

II. Algorithms and Complexity Measures

The complexity of various functions will be measured under the assumption that it is computed by a "straight-line" algorithm, which is defined below.

Definition. Let Ω be a finite set of Boolean functions

$$h_i : (\Sigma_2)^{r_i} \rightarrow \Sigma_2$$

where $\Sigma_2 = \{0, 1\}$ and $r_i \leqq r$, the fan-in of Ω. Let

$$\Gamma = \Sigma_2 \cup \{x_1, x_2, \cdots, x_N\}$$

be the *data set* where x_i is a Boolean variable. Then, a K-step *straight-line algorithm* with data set Γ is a K-tuple $\beta = (\beta_1, \beta_2, \cdots, \beta_K)$, where at the kth step $\beta_k \in \Gamma$ or $\beta_k = (h_i; k_1, \cdots, k_{r_i})$ for some $h_i \in \Omega$, where $1 \leqq k_\ell < k$ for $1 \leqq \ell \leqq r_i$. If $\beta_k \in \Gamma$, we associate with it the function $\overline{\beta}_k$, which is either a constant or a variable x_i. If $\beta_k = (h_i; k_1, \cdots, k_{r_i})$, we associate with it the recursively defined function $\overline{\beta}_k = h_i(\overline{\beta}_{k_1}, \overline{\beta}_{k_2}, \cdots, \overline{\beta}_{k_{r_i}})$. The algorithm is said to *compute* the functions $\overline{\beta}_{m_1}, \overline{\beta}_{m_2}, \cdots, \overline{\beta}_{m_q}$ if $\beta_{m_1}, \beta_{m_2}, \cdots, \beta_{m_q}$ are steps of the algorithm. The set of primitives Ω is said to be *complete* if every Boolean function

$$f : (\Sigma_2)^N \rightarrow \Sigma_2$$

can be computed by some straight-line algorithm over Γ.

Associated with an algorithm β is a *graph* of the algorithm. This is a set of vertices that are in a 1-1 correspondence with the steps of β and a set of directed edges. A step $\beta_k \in \Gamma$ corresponds to a vertex which has no edges directed into it while a step $\beta_k = (h_i; k_1, \cdots, k_{r_i})$ has a corresponding vertex with ordered edges directed into it from nodes corresponding to $\beta_{k_1}, \cdots, \beta_{k_{r_i}}$. A graph is said to have *fan-out* of s if the maximum number of edges directed away from any vertex is s. This is also said to be the fan-out of the algorithm. If $\beta_k \in \Gamma$, its corresponding node is called a *source vertex*. The graph of an algorithm is also known as the circuit diagram of a *combinational machine*.

A cost, P_i, is associated with the use of each primitive operation $h_i \in \Omega$ and the cost of an algorithm is the sum of the costs of each of its primitive operations. Then, the *combinational complexity* with fan-out s of the Boolean functions $f_1, f_2, \cdots, f_L$ with respect to the set of primitives Ω, $C_s(f_1, f_2, \cdots, f_L)$, is the smallest cost of any straight-line algorithm over Ω which computes these functions and which has fan-out of at most s. Then, it is clearly true that

$$C_\infty(f_1, \cdots, f_L) \leqq C_s(f_1, \cdots, f_L) \leqq C_{s-1}(f_1, \cdots, f_L) \leqq C_1(f_1, \cdots, f_L)$$

The two measures of greatest interest in this article are C_∞, the combinational complexity with unlimited fan-out, and C_1, the combinational complexity with fan-out of 1. It should be noted that an algorithm with fan-out 1 has no memory in the sense that any intermediate functions which are used more than once must be recomputed.

Lupanov (Refs. 2 and 3) has shown that every Boolean function f of N variables can be realized with

$$C_\infty(f) \leqq \rho \frac{2^N}{N}(1 + \epsilon)$$

$$C_1(f) \leqq \rho \frac{2^N}{\log_2 N}(1 + \epsilon)$$

for $0 < \epsilon$ and $N \geqq N^*(\epsilon)$, where

$$\rho = \max_{r_i \geqq 2} \frac{P_i}{r_i - 1}$$

These bounds are sharp in the sense that for large N, almost all Boolean functions have $C_\infty(f)$ and $C_1(f)$ which are larger than $\rho(2^N/N)(1 - \epsilon)$ and $\rho(2^N/\log_2 N)(1 - \epsilon)$, respectively.

Let the length of an algorithm be the number of edges on a longest directed path of its graph. Then, another important measure of complexity is the *computation time of Boolean functions* $f_1, \cdots, f_L$ denoted $D(f_1, \cdots, f_L)$, which is defined as the minimal length of a straight-line algorithm over Ω which computes $f_1, \cdots, f_L$. We have immediately that

$$D(f_1, \cdots, f_L) = \max_{1 \leqq t \leqq L} D(f_t)$$

It can be shown, using the disjunctive normal form decomposition of a Boolean function, that every Boolean function of N variable can be realized with $D(f) \leq N - 1 + \lceil \log_2 N \rceil$. Also, for large N, almost all of these functions has $D(f)$ which is bounded below by a function linear in N.

The following lemma establishes a relation between the measures D and C_1.

Lemma 1. *Let f be a Boolean function and let f be realized by a straight-line algorithm with fan-in r. Then,*

$$D(f) \geq \left\lceil \log_r \left[(r-1) \frac{C_1(f)}{P_{\max}} + 1 \right] \right\rceil$$

where $P_{\max}$ is the maximum cost P_i for $h_i \in \Omega$.

Proof. Consider an algorithm which realizes f with computation time $D(f)$. Such an algorithm can be assumed to have fan-out of 1 and to have an associated graph which is a tree since only one vertex of the graph corresponds to the function f which is to be computed. Then, the graph of such an algorithm cannot have more than $(r^{D(f)} - 1)/(r - 1)$ non-source vertices and this quantity must be at least as large as $C_1(f)/P_{\max}$. The inequality then follows.

Q.E.D.

A test is now developed which when satisfied provides a linear lower bound to combinational complexity.

Lemma 2. *Let f be a Boolean function which is dependent on N variables. Then,*

$$C_s(f) \geq \rho(N-1), \qquad 1 \leq k \leq \infty$$

where

$$\rho = \min_{r_i \geq 2} \frac{P_i}{r_i - 1}$$

and r_i is the number of inputs on which h_i depends.

Proof. Let β be an optimal algorithm of fan-out s which computes f. Then, the graph of the algorithm has exactly one vertex with no edges directed away from it with all remaining vertices having at least one edge directed away from them. Thus, the number of edges directed away from vertices is at least

$$N - 1 + \sum_i M_i$$

where M_i is the number of occurrences of h_i since each of the N variables on which f depends are associated with source vertices. But there are at most

$$\sum_i M_i r_i$$

edges directed into vertices so

$$N - 1 \leq \sum M_i (r_i - 1)$$

If i_0 minimizes $P_i/(r_i - 1)$, then

$$\rho(N-1) \leq \sum_i M_i \frac{P_{i_0}}{(r_{i_0} - 1)} (r_i - 1)$$

$$\leq \sum_i M_i P_i = C_s(f)$$

which is the desired result.

Q.E.D.

Any function satisfying this test has a combinational complexity which grows at least linearly with the number of variables on which it depends. For $s > 1$, this is the strongest known lower bound. For $s = 1$, Specker's Theorem (Ref. 4) states conditions under which $C_1(f)$ must grow faster than linearly in N, N being the number of variables. In the next section we present a technique which we call Neciporuk's test for generating lower bounds to $C_1(f)$. These bounds can grow as fast as $N^2/\log N$.

A few remarks are in order concerning straight-line algorithms. These algorithms contain no loops nor do they permit conditional branching. For this reason they in general cannot compute functions as quickly as algorithms which have these features. Nevertheless, results of Savage (Ref. 5) show that if A is an autonomous sequential machine which allows loops and conditional branches and computes f, then the inequality $C_\infty(f) \leq (C^*_\infty(A) + 5n_0)T$ must be satisfied, where T is the maximum number of cycles executed by A on any point in the domain f, $C^*_\infty(A)$ is the combinational complexity of the next state and output function of A, and n_0 is the number of binary digits required to specify the output of A. We can assert, for example, that if T is much smaller than $C_\infty(f)$, then the complexity of the machine A must be large.

III. Neciporuk's Test

In this section we develop a lower bound to $C_1(f)$ for a Boolean function of N variables. This bound was abstracted from Neciporuk's demonstration (Ref. 6) that for a certain function (actually a sequence of functions) $C_1(f)$ grows as fast as $N^2/\log N$, N being the number of variables on which f depends.

Given a Boolean function f of N variables $x_1, \cdots, x_N$, let

$$f^{x_{i_1}, \cdots, x_{i_{N-m}}}_{a_1, \cdots, a_{N-m}} \Big| _{x_{j_1}, \cdots, x_{j_m}}$$

denote the function of $m \leqq N$ variables derived from f by setting x_{i_K} to the constant a_K. The number of distinct such functions is bounded by 2^{2^m} and 2^{N-m}.

Theorem. *Suppose f is a Boolean function dependent on N variables, N being divisible by m. If the variables can be partitioned into N/m blocks B_i containing m elements each, in such a way that there are F distinct functions*

$$f^{x_j \notin B_i}_{a_j} \Big| _{x_j \in B_i}$$

for each i, then

$$C_1(f) \gtrsim \frac{\rho}{3(r-1)} (N/m) \log_2 F$$

where $r = \max r_i$.

Proof.

(i) Adding functional elements to the basis Ω cannot increase $C_1(f)$. The element we shall add has three inputs, x, η, and γ, and computes the function $f(x, \eta, \gamma) = \eta x \oplus \gamma$, $\oplus$ denoting EXCLUSIVE OR. Let P_i for this element be 2ρ so that $P_i/(k_i - 1) = \rho$ and this parameter is not lowered for Ω', the enlarged basis. Then $C_1(f) \geqq C_1'(f)$.

(ii) Let β be an optimal straight-line algorithm over Ω', with fan-out one, which computes f. Let t_i be the number of inputs of variables from block B_i and $t = \Sigma t_i$, the total number of inputs. By the linear inequality of Lemma 2,

$$C_1'(f) \geqq \rho(t-1) \simeq \rho t$$

(iii) If i_0 is the index for which

$$t_{i_0} = \min_i t_i$$

then $t \geqq (N/m) t_{i_0}$.

(iv) Consider β as computing the function

$$f^{x_j \notin B_{i_0}}_{a_{i_j}} \Big| _{x_j \in B_{i_0}}$$

by setting the $x_j \notin B_{i_0}$ to constants (call them *free constants*). β may be far from optimal for this task, however, and the following operations might improve it: If any subtree of the graph of β has only constant or free constant inputs, replace that tree by a node representing a new free constant. If any subtree has exactly one variable input, then replace it by the element computing $\eta x \oplus \gamma$, with x being the variable input and η, γ free constants. The desirable property of this element is that

$$f^{\eta\gamma}_{a_1 a_2} \Big| _x$$

can be any function of x, depending on the values of a_1 and a_2. Call the algorithm derived from β by the above transformations β_0. β_0 will still compute any function

$$f^{x_j \notin B_{i_0}}_{a_j} \Big| _{x_j \in B_{i_0}}$$

if the proper values of its free constants are given. Also, it still has t_{i_0} variable inputs.

(v) β_0 has no elements without at least one variable input and no chains of elements with only one variable input to the chain. If we overlook the elements with only one variable input, then all other elements have at least two variable inputs and a simple induction shows that t_{i_0} is at least one more than the number of such elements. If a_1 is the number of elements with one variable input, a_2 is the number of elements with at least two variable inputs, and C_0 is the total number of functional elements, then $C_0 = a_1 + a_2$, and from the above $t_{i_0} \geqq a_2 + 1$. Finally, each element with at least two variable inputs and the inputs of $x_j \in B_{i_0}$ can be followed by at most one element with a single variable input and so $a_2 + t_{i_0} \geqq a_1$. Therefore, $C_0 < 3t_{i_0}$.

(vi) By another application of Lemma 2, this time with $P_i = 1$ for all functional elements,

$$C_0 \geqq \frac{F_0 - 1}{r - 1} \simeq \frac{F_0}{r - 1}$$

where F_0 is the number of free constants in β_0.

(vii) Recalling that β_0 computes at least F distinct functions depending on the choice of values for free constants, $F_0 \geqq \log_2 F$.

Now, putting all of these inequalities together,

$$\begin{aligned}
C_1(f) &\geqq C_1'(f) && \text{(i)}\\
&\gtrsim \rho t && \text{(ii)}\\
&\geqq \rho (N/m)\, t_{i_0} && \text{(iii)}\\
&\gtrsim \rho (N/m)(C_0/3) && \text{(v)}\\
&> (\rho/3)(N/m)(F_0/r - 1) && \text{(vi)}\\
&\geqq [\rho/3(r-1)](N/m)\log_2 F && \text{(vii)}
\end{aligned}$$

As an example of how Neciporuk's test works, we can exhibit Neciporuk's function: Given a positive integer N, let $m = 2 + \log_2 N$ and let X be an $[N/m] \times m$ matrix of independent Boolean variables, x_{ij}. Also let σ be an $[N/m] \times m$ matrix of distinct m-vectors of 0's and 1's, each containing at least three 1's. Then,

$$\mathcal{N}(X) = \sum_{ij} x_{ij} \sum_{\substack{k \\ (k \neq i)}} \prod_{\sigma_{ij\ell} = 1} x_{k\ell}$$

where $\sum$ means repeated application of "exclusive or" (sum mod 2). The blocks of the partition in Neciporuk's test will be the rows of X. Fixing i, if $a_{h_0 j_0} \neq a'_{h_0 j_0}$ for some $h_0 \neq i$, then

$$\mathcal{N}^{x_{hj},\, h\neq i}_{a_{hj}} \Big|_{x_{i1},\,\cdots,\,x_{i,m}} \neq \mathcal{N}^{x_{hj},\, h\neq i}_{a'_{hj}} \Big|_{x_{i1},\,\cdots,\,x_{i,m}}$$

since one of them contains the term

$$\prod_{\sigma_{h_0 j_0 \ell} = 1} x_{i\ell}$$

and the other does not. Therefore,

$$C_1(\mathcal{N}) \gtrsim \frac{\rho}{3} \frac{N^2}{\log_2 N}$$

the strongest lower bound possible from Neciporuk's test.

On the other hand the definition of $\mathcal{N}$, above, gives

$$C_1(\mathcal{N}) \lesssim \frac{N^2}{\log_2 N} P_{\oplus} + \frac{N^2}{2} P_{\cdot} \simeq \frac{N^2}{2} P_{\cdot}$$

showing that Neciporuk's test is fairly accurate.

IV. The Marriage Problem and the Traveling Salesman's Problem

These problems are a familiar part of combinatorial theory [see Berge (Ref. 7), Gomory (Ref. 8), Harper–Rota (Ref. 9), Mirsky–Perfect (Ref. 10), and Lin (Ref. 11)]. However, since some variation is possible, let us begin with precise definitions: Let X be an arbitrary real $n \times n$ matrix and

$$m(X) = \max_{P} X \cdot P$$

where P ranges over all $n \times n$ permutation matrices and

$$X \cdot P = \sum_{i,j} x_{ij} p_{ij}$$

Then find a P such that $X \cdot P = m(X)$. This is generally called the *marriage problem*, but, for reasons which will become apparent, we shall be more interested in the closely related problem of finding a simple algorithm to compute $m(X)$.

The marriage problem is so called because one may imagine a population of n boys and n girls with x_{ij} being the happiness generated if boy i marries girl j. The problem then is to arrange the n marriages so that the total happiness is maximized.

The value of m at X may always be computed by evaluating $X \cdot P$ for each permutation matrix P and comparing. But this requires $n!$ separate operations which quickly surpasses the capabilities of even our largest and fastest machines. There are more efficient algorithms known, however. As we show in *Section V*, the classical alternating paths algorithm requires on the order of n^4 elementary operations. Hopcroft and Karp have in private communications indicated that this figure may be lowered to $n^{2.5}$ if conditional branch operations are allowed.

The *traveling salesman's problem*, with the same notation, is to find a *cyclic* permutation matrix, P, minimizing $X \cdot P$. As before we shall investigate algorithms for computing

$$t(X) = \min_{P \text{ cyclic}} X \cdot P$$

x_{ij} may now be thought of as the cost of traveling from city i to city j and a traveling salesman of course would wish to make a complete tour at minimal total cost. Again the trivial solution would entail evaluating $X \cdot P$ over all $(n-1)!$ cyclic permutation matrices ($n! \sim \sqrt{2\pi n}\,(n/e)^n$ by Sterling's formula). Bellman [see Gomory (Ref. 5)] has given a dynamic program which reduces this to approximately $n^2 \cdot 2^n$ operations, still exponentially increasing. We shall pass over the clever work on partial solutions and solutions for special cases by saying that no one has yet to our knowledge found an algorithm for computing $t(X)$ which does a number of computations growing like n^k for fixed k.

In any case the traveling salesman's problem is at least as complex (modulo a small constant) as the marriage problem since the latter can be embedded in the former. If all the entries of X are positive, and we may assume this since adding a constant to all entries of X does not significantly alter the problem, then

$$m(X) = -t\begin{pmatrix} 0 & -X \\ 0 & 0 \end{pmatrix}$$

where the argument of t is $2n \times 2n$. Any lower bound we obtain for the complexity of m then gives a lower bound of the same order for the complexity of t.

In order to apply Neciporuk's test to the marriage problem, we restrict the x_{ij} to values 0 and 1 and let $m_0(X) \equiv m(X) \pmod 2$. Clearly, neither of these transformations would significantly increase the complexity of computing any function, and one may argue that in fact the essential complexity of the marriage problem is retained in the Boolean function $m_0(X)$.

Now X is a matrix of $N = n^2$ Boolean variables, so let $m = n$ and $B_K = \{x_{ij} : 1 \leq i,\ j \leq n,\ i \equiv j + K \pmod n\}$, $K = 0, 1, \cdots, n-1$. The B_K's partition X and any B_K may be mapped onto any other by row and column permutations which do not effect $m(X)$. F is then the number of distinct functions

$$m_0 {}^{x_{ij} \notin B_0}_{a_{ij}} \Big|_{x_{ij} \in B_0}$$

where

$$B_0 = \{x_{ii}, i = 1, \cdots, n\}$$

We wish to find a good lower bound on the number of distinct functions

$$m_0 {}^{x_{ij},\, i \neq j}_{a_{ij}} \Big|_{x_{11}, \cdots, x_{nn}}$$

There are $2^{n^2 - n}$ such functions and we shall show that at least $2^{n^2/4}$ of them are distinct if n is even. This is established by the following claim: If n is even, then the functions

$$m_0 {}^{x_{ij},\, i \neq j}_{a_{ij}} \Big|_{x_{11}, \cdots, x_{nn}}$$

for which $a_{ij} = 0$ unless $i \leq n/2 < j$ are all distinct.

Proof of claim. Suppose a_{ij} and a'_{ij} $(i \neq j)$ determine two such functions and that for

$$i_0, j_0, i_0 \leq \frac{n}{2} < j_0$$

$$a_{i_0 j_0} = 0$$

and

$$a'_{i_0 j_0} = 1$$

Then,

$$m {}^{x_{ij},\, i \neq j}_{a_{ij}} \Big|_{x_{11}, \cdots, x_{nn}} = n - 2$$

at the point

$$x_{ii} = \begin{cases} 1, \text{ if } i \neq i_0, j_0 \\ 0, \text{ if } i = i_0 \text{ or } j_0 \end{cases}$$

and

$$m {}^{x_{ij},\, i \neq j}_{a'_{ij}} \Big|_{x_{11}, \cdots, x_{nn}} = n - 1$$

at the same point. Therefore,

$$m_0 {}^{x_{ij},\, i \neq j}_{a_{ij}} \Big|_{x_{11}, \cdots, x_{nn}} \neq m_0 {}^{x_{ij},\, i \neq j}_{a'_{ij}} \Big|_{x_{11}, \cdots, x_{nn}}$$

Thus, we have:

Theorem. *If X is an $n \times n$ matrix of Boolean variables,*

$$m(X) = \max_P X \cdot P$$

P ranging over all permutation matrices, and $m_0(X) \equiv m(X) \pmod 2$, then

$$C_1(m_0) \gtrsim \frac{\rho n^3}{12(r-1)}$$

Since $m_0(X)$ is dependent on all n^2 variables of X, Lemma 2 shows that $C_s(m_0) \geq \rho(n^2 - 1)$. Thus, the Neciporuk test provides a distinct improvement on this bound when $s = 1$. Lemma 1 can be used with either bound to $C_1(m_0)$ to lower bound the computation time $D(m_0)$. The bound provided by Lemma 2 shows that $D(m_0)$ must grow as $2 \log_r n$ while that of the theorem shows that $D(m_0)$ must grow as $3 \log_r n$, which is a small improvement.

To date we have been unsuccessful in generating upper bounds to $C_1(m_0)$ which are algebraic in n or upper bounds to $D_1(m_0)$, which grow less rapidly than n^2. We now state a straight-line algorithm of fan-out $s = n + 1$

which implements the "alternating path" solution to the marriage problem [Berge (Ref. 7)]. This algorithm has fan-in of $r = 2$ and uses the primitive operations of AND, OR, EXCLUSIVE OR (modulo 2 addition), and NEGATION, denoted $.$, $+$, $\oplus$, $\overline{\quad}$, respectively. The variables of X are denoted by x_{ij}, $1 \leqq i, j \leqq n$ and the intermediate variables $u_i(r,k)$, $u'_i(r,k)$, $u''_i(r,k)$, $1 \leqq i \leqq n$, $1 \leqq r \leqq k \leqq n$; $v_j(r,k)$, $v'_j(r,k)$, $v''_j(r,k)$, $1 \leqq j \leqq n$, $1 \leqq r \leqq k \leqq n$; $\gamma_{ij}(k)$, $1 \leqq i, j, k \leqq n$, and $\theta_j(k)$, $1 \leqq j, k \leqq n$ are used in the description of the algorithm. If any of these intermediate variables are used with values for their indices which are outside the stated ranges, they are assumed to have value 0.

The intermediate functions computed by the algorithm follow:

$$u_i(1,k) = \begin{cases} 1, & i = k \\ 0, & i \neq k \end{cases}$$

$$v_j(r,k) = \sum_{i=1}^{n} x_{ij} \cdot (u_i(r,k) + u_i(1,k))$$

$$u_i(r,k) = \sum_{j=1}^{n} \gamma_{ij}(k-1) \cdot v_j(r-1,k), \text{ for } r \geqq 2$$

$$\theta_j(k) = v_j(k,k) \cdot \left(\overline{\sum_{i=1}^{n} \gamma_{ij}(k-1)}\right) \cdot \left(\overline{\sum_{l=1}^{j-1} \theta_l}\right)$$

$$v'_j(r,k) = \theta_j(k) \cdot (v_j(r,k) \oplus v_j(r-1,k)) + \sum_{i=1}^{n} \gamma_{ij}(k-1) \cdot u'_i(r+1,k)$$

$$u'_i(r,k) = \left(\sum_{j=1}^{n} v'_j(r,k) \cdot x_{ij}\right) \cdot u_i(r,k) \cdot \left(\overline{\sum_{l=1}^{i-1} u'_l(r,k)}\right)$$

$$\gamma_{ij}(k) = \sum_{r=1}^{k} u'_i(r,k) \cdot v'_j(r,k) + \left(\gamma_{ij}(k-1) \oplus \sum_{r=1}^{k} u'_i(r+1,k) \cdot v'_j(r,k)\right)$$

$$m_0(X) = \sum_{i,j} \gamma_{ij}(n)$$

where Σ and Σ denote the OR and EXCLUSIVE OR, respectively, of more than 2 terms. If P_-, $P_.$, P_+, and $P_\oplus$ are the costs of the NEGATION, AND, OR, and EXCLUSIVE OR operations, then the cost of this algorithm is

$$C = \left(\frac{n^3}{2} + \frac{n^2}{2} + 2n\right) P_- + \left(3n^4 + \frac{5n^3}{2} + 8n^2\right) P_. + \left(\frac{7n^4}{2} - n^3 - 3n^2 + n\right) P_+ + \left(\frac{3n^3}{2} + \frac{n^2}{2} - 1\right) P_\oplus$$

Since this algorithm has fan-out of $s = n + 1$, we have

$$C_\infty(m_0) \leqq C_{n+1}(m_0) \leqq C_1$$

and this bound grows as n^4 for large n.

V. Conclusion

We have distilled on these pages the main thrust and concrete results which grew out of discussions between the two authors. Side issues and philosophy have been avoided even though they were an integral part of the development of our thoughts. We should like to state here, however, our conviction that combinational complexity (particularly with unlimited fan-out) is a fundamental concept in the study of combinatorial optimization problems. Questions such as those at the beginning of *Section I* assume a precise meaning in these terms and can be treated rationally. The old bugaboos about trade-offs between computing time, memory (random access, tape, multiple tape, etc.), and other machine characteristics are resolved. Definitive statements may even be made about machines capable of looping and conditional branch operations. The key to these applications of combinational complexity is the ability of combinational machines (or equivalently, straight-line algorithms) to model the components and operations of all other digital machines. For details the reader is referred to the papers of Savage (Ref. 5) on combinational complexity (also see Ref. 1).

The results of this article are mainly interesting in that they suggest avenues of further research and we should like to point out those which in our opinion are the most promising: (1) To what other "natural" functions can Neciporuk's test be applied? (2) One of the continuing paradoxes of this subject is that according to Lupanov almost all functions of N variables have

$$C_l(f) \simeq \rho \frac{2^N}{\log N}$$

and yet Neciporuk's function N with

$$C_1(N) \gtrsim \rho \frac{N^2}{\log N}$$

is the world's champion among those explicitly given. Can Neciporuk's technique be generalized to give stronger lower bounds, for instance $C_1(f) \geqq N^K$, for all $K \geqq 1$?

Interesting candidates for large complexity are not lacking. Besides the traveling salesman's function, we have the closely related combinatorial coding function [see Harper (Ref. 12)], and many functions from graph theory such as the chromatic number, χ_G. If G is a graph whose incidence matrix is symmetric $n \times n$ matrix of 0's and 1's (so $N = n(n+1)/2$), then χ_G is the minimum number of colors which can be used to color the vertices of G so that no two neighboring vertices have the same color (see Ref. 7). Another candidate which we cannot resist mentioning is the complexity function itself. For any Boolean function f of n variables, there are 2^n points in the domain. If these points are ordered in their natural binary order, say, then any function of n variables is equivalent to a binary sequence of length 2^n. $C_K(f)$ can then be looked upon as a function of $N = 2^n$ variables (f representing a point in the domain). Thus, we can inquire about the complexity of complexity and it seems reasonable to attribute the paradoxes in our theory to the fact that complexity is most likely a very complex function.

References

1. Savage, J. E., "Digital Telemetry and Command: A Collection of Results on Computational Complexity," in *The Deep Space Network*, Space Programs Summary 37-65, Vol. II, pp. 42–47. Jet Propulsion Laboratory, Pasadena, Calif., Sept. 30, 1970.
2. Lupanov, O. B., "On a Method of Machine Synthesis," *Izvestia B.Y.Z., Radiofizika,* Vol. 1, pp. 120–140, 1959.
3. Lupanov, O. B., "On the Difficulty of Realizing Functions of Logical Algebra with Formulas," *Sb. Problemi Kibernetiki, Fitzmatgiz,* Vol. 3, pp. 61–80, 1960.
4. Hodes, L., and Specker, E., "Lengths of Formulas and Elimination of Quantifiers," in *Contributions to Mathematical Logic,* pp. 175–188. Edited by K. Schutte. North Holland, Amsterdam, 1968.
5. Savage, J. E., *Computational Work and Time on Finite Machines,* 1971 (to be published).
6. Neciporuk, E. E., "A Boolean Function," *Soviet Math-7, Doklady 7,* pp. 999–1000, 1966.
7. Berge, C., *The Theory of Graphs and Its Applications.* John Wiley & Sons, Inc., New York, 1964.
8. Gomory, R. E., "The Traveling Salesman Problem," in *Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems,* pp. 93–121. IBM, White Plains, New York, 1966.
9. Harper, L. H., and Rota, G-C., "Matching Theory, An Introduction," in *Advances in Probability Theory,* Vol. 1, pp. 172–215. Marcel Dekker, New York, 1971.
10. Mirsky, L., and Perfect, H., "Systems of Representatives," *J. Mathematical Analysis and Applications,* Vol. 15, No. 3, pp. 520–568, Sept. 1966.
11. Lin, S., "Computer Solutions of the Traveling Salesman Problem," *Bell System Tech. J.,* Vol. 44, pp. 2245–2269, 1965.
12. Harper, L. H., "The Combinatorial Coding Problem," in *Proceedings of the Second Chapel Hill Conference on Combinatorial Mathematics and Its Applications,* Chapel Hill, North Carolina, pp. 252–260, 1970.