AN EXAMPLE OF INFORMATION AND COMPUTATION RESOURCE TRADE-OFF

by

R. P. Daley

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania

May, 1971

ABSTRACT

This paper is concerned with establishing the existence of a recursive sequence x which exhibits a "continuous" trade-off between the length of programs (information) which compute the finite initial segments of x and the computation resources (e.g. computation time) required by these programs for such computations.

1. INTRODUCTION

In an attempt to measure the information content of finite and infinite binary sequences, Kolmogorov [6], Chaitin [3,4] and Loveland [8] have formulated minimal-program complexity measures. The main result of this paper, which is presented in the following section, shows the existence of a recursive sequence x which exhibits a "continuous" trade-off between the length of programs (information) which compute the finite initial segments of x and the computation resources (e.g., computation time) required by these programs for such computations. In this section we present the basic definitions and properties of the complexity measures needed for the establishment of such results.

If x is a binary sequence (finite or infinite) then we denote by $x(n)$ the n^{th} member of x and by x^n the initial segment of x of length n, i.e. $x^n = x(1)\cdots x(n)$. If x is a binary string (finite sequence) then we denote by $|x|$ the length of x, i.e. the number of symbols of x. We say that the string y extends the string x, which we denote by $x \prec y$, if and only if $\forall i \leq |x| . x(i) = y(i)$.

For this paper we will use Loveland's definition of minimal-program complexity. The minimal-program complexity of x^n is defined by,

$$K_A(x^n;n) = \min \{|p| \quad | p \text{ is a binary string and } \forall i \leq n . A(p,i) = x^i\},$$

where A is an algorithm.

$$= \infty \text{ if no such } p \text{ exists}$$

One may regard A as a digital computer and p as the encoding of a computer program which when run on A with input i produces x^i as output. Thus p contains the information and procedure necessary for the computation of x^i and so, intuitively, $K_A(x^n;n)$ measures the information (other

than n) required to compute x^n. Loveland's definition is a variant of Kolmogorov's intended to insure that the only information provided by n to the program computing x^n is that n is the length of x^n. Many of the basic properties of this minimal-program complexity measure are discussed in detail in [8,9].

By the same method that Kolmogorov used for his formulation of minimal-program complexity measure one can show the existence of a "universal" algorithm A such that for any algorithm B there is a constant c such that $\forall x \forall n . K_A(x^n;n) \leq K_B(x^n;n)+c$. We will fix such a universal algorithm A and therefore delete the subscript. By "$\overset{\infty}{\exists} n.$" and "$\overset{\infty}{\forall} n.$" we mean "there exist infinitely many n such that" and "for all but finitely many n" respectively. We define (following [9]) the complexity class named by f by

$$C[f] = \{x \mid \overset{\infty}{\forall} n . K(x^n;n) \leq f(n)\}$$

Since we designate complexity classes by the names of functions we will use λ notation to specify the name of a function in terms of its values (e.g. $[\lambda n . n^2]$ is the name of the function f such that $f(n)=n^2$). We will denote the greatest integer $\leq n$ by $[n]$.

We now present some basic properties of this complexity measure which are relevant to the results in the following section.

1. Every binary sequence x has a well defined complexity, indeed there is a constant c_o such that $\forall x . x \in C[\lambda n . n+c_o]$.

2. A sequence x is recursive if and only if there exists a constant c such that $x \in C[\lambda n . c]$.

3. If x is a recursively enumerable sequence (i.e. the characteristic sequence of a recursively enumerable set) then there is a constant c such that $x \in C[\lambda n . \log_2(n)+c]$. (See [1] and [9]).

Kolmogorov first suggested considering the minimal-program complexity of sequences when the computation resources used by programs computing their initial segments are restricted, and Barzdin [1] first utilized this idea. In a similar manner, we employ this idea and obtain a related complexity measure which we will refer to as the bounded complexity measure. Let $[\lambda p.\langle p\rangle]$ be an effective bijection between the set X^* of all finite binary sequences and the set N of natural numbers. We will enumerate all 0,1-valued partial recursive functions in the following manner: for every $p\in X^*$ and every $n\in N$, $\varphi_{\langle p\rangle}(n)=A(p,n)(n)$, i.e. $\varphi_{\langle p\rangle}(n)$ is the n^{th} member of the string $A(p,n)$. Let $\Phi=\{\Phi_i\}$ be any Blum complexity measure for $\{\varphi_i\}$ (see [2]). Thus Φ satisfies 1) $\varphi_i(n)$ is defined if and only if $\Phi_i(n)$ is defined, and 2) the predicate $\Phi_i(n)\leq m$ is recursive.

We define the bounded (with bound t) minimal-program complexity of x^n by,

$$K^t_\Phi(x^n;n) = \min\{|p| \mid p \text{ is a binary string and } \forall i\leq n(A(p,i)=x^i \text{ and } \Phi_{\langle p\rangle}(i)\leq t(i))\}$$

Intuitively, $K^t_\Phi(x^n;n)$ measures the information (other than n) required to compute x^n within t(n) units of the computation resource Φ. We introduce the following complexity classes,

$$C_\Phi[f|t]=\{x|\overset{\infty}{\forall} n.K^t(x^n;n)\leq f(n)\}$$

$$C^{bd}_\Phi[f]=\{x|\overset{\infty}{\forall} n.K^t(x^n;n)\leq f(n), \text{ for some total recursive } t\}$$

We will fix a Blum complexity measure Φ and therefore will delete the subscript Φ. Corresponding to 1 and 2 above we have the following,

1′. There is a constant c_o and a total recursive function t_o such that $\forall x.x\in C[\lambda n.n+c_o|t_o]$.

2.′ A sequence x is recursive if and only if there is a constant c such that $x \in C^{bd}[\lambda n.c]$.

It will be useful to consider the following variant of the bounded complexity,

$$^{*}K^{t}_{\Phi}(x^{n};n) = \min \{ |p| \mid p \text{ is a binary string and } \Phi_{<p>}(n) \leq t(n) \text{ and } \forall i \leq n.A(p,i) = x^{i} \}.$$

Clearly, $^{*}K^{t}_{\Phi}(x^{n};n) \leq K^{t}_{\Phi}(x^{n};n)$. It is also the case that all the theorems in this paper which are stated for $K^{t}_{\Phi}(x^{n};n)$ are also true for $^{*}K^{t}_{\Phi}(x^{n};n)$. We denote the complexity classes for this measure by $^{*}C_{\Phi}[f|t]$ and $^{*}C^{bd}_{\Phi}[f]$. The essential difference between these two measures is that for $^{*}K^{t}_{\Phi}(x^{n};n)$ we restrict the computation resources used only for the computation of x^{n} (there is <u>another</u> program which computes x^{i} in t(i) units). $^{*}K^{t}_{\Phi}(x^{n};n)$ still insures, as does $K^{t}_{\Phi}(x^{n};n)$, that n indicates only the length of the sequence to be computed.

2. INFORMATION -- COMPUTATION RESOURCE TRADE-OFF

Motivation for investigating the trade-off between program length and computation resources is provided by the following result (see [5]).

3'. There exists a recursively enumerable sequence x such that for every constant $c < 1$, $x \notin C^{bd}[\lambda n.c \cdot n]$.

Thus in view of 3 (see section 1) the information requirements for such a sequence increase drastically whenever an effective bound is placed on the computation resources for its computation. We mention that Barzdin [1] previously proved a similar result for the time of computation measure Φ: there is a recursively enumerable sequence x such that for every total recursive function t there is a constant $c_t < 1$ such that $\forall n.K_{\Phi}^{t}(x^n;n) \geq c_t \cdot n$. The difference between Barzdin's result and 3' is mainly one of quantification:

$$\forall t \exists c < 1 \forall n.K^t(x^n;n) > c \cdot n \text{ and } \forall t \forall c < 1 \exists^{\infty} n.K^t(x^n;n) > c \cdot n.$$

The following theorem and especially its corollary show that there is a sequence x which exhibits a "continuous" trade-off between information requirements and other computation resource requirements for the computation of its initial segments, i.e., as we decrease the amount of information provided in a program computing x^n we must increase the amount of computation resource used in the computation by such a program. We say that a <u>sequence of functions $\{f_i\}$ is recursive</u> (primitive recursive) if and only if the function $F(i,n) = f_i(n)$ is total recursive (primitive recursive).

Theorem 1: If $\{f_i\}$ and $\{t_i\}$ are recursive sequences of functions satisfying (1) $\forall i.\ f_i$ is unbounded non-decreasing, (2) $\forall n.\ f_i(n) \leq n$, and (3) $\forall i \forall n. f_{i+1}(n) \leq \frac{1}{3} \cdot f_i(n)$, then there is a recursive sequence x and a recursive sequence of functions $\{s_i\}$ such that (1) $\forall i > 1.\ x \notin C[f_i | t_i]$ and (2) $\forall i.\ x \in C[f_i | s_i]$.

Before we present the proof, we will discuss briefly the statement of Theorem 1. The number $\frac{1}{3}$ (rather than $\frac{1}{2}$) in condition (3) seems to be essential to the construction in the general case. However, we are able to give examples where the $\{f_i\}$ satisfy the weaker condition $\forall i \forall n. f_{i+1}(n) \leq \frac{1}{2} \cdot f_i(n)$. The trade-off is more evident in the following corollary to the proof of Theorem 1, obtained by letting $t_{i+1} = s_i$ in Theorem 1.

Corollary: If $\{f_i\}$ is a recursive sequence of functions satisfying (1) $\forall i.\ f_i$ is unbounded non-decreasing, (2) $\forall n.\ f_i(n) \leq n$, and (3) $\forall i \forall n.\ f_{i+1}(n) \leq \frac{1}{3} \cdot f_i(n)$, then there is a recursive sequence x and a recursive sequence $\{t_i\}$ of functions such that (1) $\forall i > 1$, $x \notin C[f_i | t_i]$ and (2) $\forall i.\ x \in C[f_i | t_{i+1}]$.

The $\{f_i\}$ is a decreasing sequence of information bounds and the $\{t_i\}$ is clearly an increasing sequence of computation resource bounds. Simply stated, x is a sequence such that as we decrease the amount of information provided for the computation of x^n we must increase the amount of computation resources permitted for the computation of x^n. The Corollary becomes even more interesting in view of 2' which states that since x is recursive there is a constant c and a total recursive function t such that $x \in C[\lambda n.c | t]$.

Proof: (of Theorem 1). We construct x in stages. At each stage n we define x(j) for $j \in (e_{n-1}, e_n]$ where e_n is an appropriately defined recursive function of n. (We use (m,n] to denote the set $\{i \mid i \in N \text{ and } m < i \leq n\}$.) The construction at stage n will be carried out in n substages. At each substage m of stage n we define x(j) for $j \in (e_{n,m-1}, e_{n,m}]$ where $e_{n,0} = e_{n-1}$ and $e_{n,n} = e_n$. The construction at substage m of stage n will insure that x satisfies (1) $K^{t_m}(x^{e_n}; e_n) > f_m(e_n)$ for $m > 1$, and (2) $K^{s_m}(x^{e_n}; e_n) \leq f_m(e_{n,m}) - n + c_m$ for some constant c_m and some total recursive function s_m. We accomplish (1) by defining $x^{e_{n,m}}$ to be the least string y (with respect to the lexicographical ordering) such that $|y| = e_{n,m}$ and y extends $x^{e_{n,m-1}}$ and such that $K^{t_m}(y; |y|) > f_m(e_n)$. The existence of such a y will follow from our choice of $e_{n,m}$. It will then follow that there is a program π_m which computes x(j) for $j \in (e_{k,m-1}, e_{k,m}]$ for every $k \geq m$. We accomplish (2) simply by making $e_n - e_{n,m} < f_m(e_{n,m}) - f_m(e_{n-1})$. To do this we must use the fact that $\forall k.f_1(k) \leq k$ and $\forall i \leq n \forall k.f_i(k) \leq \frac{1}{3}f_{i-1}(k)$. Thus no matter how we later define x between $e_{n,m}$ and e_n we can always compute x^{e_n} by specifying x between $e_{k,m}$ and e_k for $m \leq k \leq n$ in addition to the program π_m^* which computes x between e_{k-1} and $e_{k,m}$ for $m \leq k \leq n$ by using the programs π_j for $j \leq n$. Hence for some total recursive function s_m and some constant c_m,

$$K^{s_m}(x^{e_n}; e_n) \leq \sum_{k=1}^{n} (f_m(e_{k,m}) - f_m(e_{k-1}) - 1) + c_m$$

$$\leq f_m(e_{n,m}) - n + c_m$$

We remark that although π_m^* requires n binary strings to compute x^{e_n}, the

above inequality is obtained since the length of each of these strings is known to π^*_m and so we can compute x^{e_n}, using π^*_m, from the concatenation of these strings.

We now present the formal details. Define $e_0=0$

and $e_{n+1} = \min \{k | k \geq e_n + \sum_{m=2}^{n} (f_m(k)+1)$ and

$\forall m \leq n \ (\sum_{j=m+1}^{n}(f_j(k)+1) < f_m(k-\sum_{j=m+1}^{n}(f_j(k)+1))-f_m(e_n))\}$.

Define $e_{n,1} = e_n - \sum_{m=2}^{n}(f_m(e_n)+1)$ and

$e_{n,m} = e_{n,m-1} + f_m(e_n)+1$.

We are guaranteed that such a k exists in the definition of e_{n+1} since $\{f_i\}$ satisfies the three conditions in the hypothesis of the theorem. In fact it is crucial for this construction that $\{f_i\}$ satisfies all three conditions. We observe also that $e_n - e_{n,m} < f_m(e_{n,m}) - f_m(e_{n-1})$.

Define $K_{n,m} = \{y^{e_n} | K^{t_m}(y^{e_n};e_n) \leq f_m(e_n)\}$.

Clearly, e_n, $e_{n,m}$ are recursive functions. Since the predicate $\Phi_i(n) \leq m$ is recursive, $K_{n,m}$ is a recursive set function. We define,

$x^{e_{n,m}} = \min \{y | \ |y|=e_{n,m}$ and $x^{e_{m,n-1}} < y$ and $y \notin K_{n,m}\}$.

Such a y exists since the cardinality of $K_{n,m}$ is less than $2^{f_m(e_n)+1}$ and $e_{n,m} - e_{n,m-1} = f_m(e_n)+1$. It follows from our previous remarks that $\forall n \forall m > 1 . K^{t_m}(x^{e_n};e_n) > f_m(e_m)$ so that $x \notin C[f_m | t_m]$, for $m > 1$. Also, we have that $\forall m \forall n . K^{s_m}(x^{e_n}; e_n) \leq f_m(e_{n,m}) - n + c_m$ and by our construction, $\forall^{\infty} m \forall n . K^{s_m}(x^n;n) \leq f_{m(n)}$ so $x \in C[f_m | s_m]$. Furthermore, $\{s_i\}$ is a recursive sequence of functions.

We show now that under certain conditions we can choose the sequence x in Theorem 1 to be primitive recursive. We say that a Blum measure Φ is primitive recursively decidable if and only if the predicate $\Phi_i(n) \leq m$ is primitive recursive.

Theorem 2: If $\{f_i\}$ and $\{t_i\}$ are primitive recursive sequences of functions satisfying (1) $\forall i.f_i$ is unbounded non-decreasing, (2) $\forall n.f_1(n) \leq n$, (3) $\forall i \forall n.f_{i+1}(n) \leq \frac{1}{3} \cdot f_i(n)$, and (4) $[\lambda i \lambda n.\ \min \{k | f_i(k) > f_i(n)\}]$ is primitive recursive, and Φ is a primitive recursively decidable Blum measure then there is a primitive recursive sequence x and a recursive sequence $\{s_i\}$ of functions such that (1) $\forall i>1.\ x \notin C_\Phi[f_i|t_i]$ and (2) $\forall i.\ x \in C_\Phi[f_i|s_i]$.

Proof: The construction is identical to that in Theorem 1. Condition (4) insures that e_n and $e_{n,m}$ are primitive recursive functions and Φ being primitive recursively decidable insures that $K_{n,m}$ is a primitive recursive set function.

We remark that the above "trade-off" theorems do not assert that a trade-off exists for all sequences, but only for some sequences. We have not established an absolute trade-off between information and computation resources, but have only demonstrated the existence of sequences which exhibit such a trade-off behavior. We conclude with some examples which demonstrate how widespread the trade-off pheonomenon is. Our discussion now will be quite informal. For these examples we will assume that our algorithm A is a processor for Markov algorithms (see [10]). Such an A is universal and we may thus restrict our programs to be Markov algorithms. Our first two examples show that the sequence of functions $\{t_i\}$ of the Corollary may be chosen so that t_{i+1} is not much larger than t_i, when $\{f_i\}$ and Φ are properly chosen. The proofs are not difficult but rather tedious so that only the statement of the example and a brief discussion will be given.

Example 1: Let Φ be the length of tape measure for Markov algorithms, i.e. Φ counts the total number of non-blank symbols on the tape at any given time

during a computation. Let $f_i(n) = [\frac{n}{2^i}]$. Since we may reuse tape during computations, we can choose the $\{t_i\}$ in the Corollary to be defined by $t_i(n)=(i+c)\cdot n$, for some constant c.

Example 2: Let Φ be the time of computation measure for Markov algorithms, i.e. Φ counts the total number of formulae applied during a computation. Let $f_i(n) = [\frac{\log(n)}{2^i}]$. Examination of the construction procedure used in Theorem 1 shows that finding $x^{e_{n,m}}$ requires at most $\sum_{i=1}^{m}(2^{f_i(e_n)+1})\cdot t_i(e_n)$ applications. It can also be shown that the remaining time needed to compute x^{e_n} by a program of length $\leq f_m(e_{n,m})$ is less than e_n^c, for some constant c. Thus we can choose the $\{t_i\}$ in the Corollary to be defined by $t_i(n) = n^{(i+c)}$, for some constant c.

We now show that under certain circumstances we may choose the sequence x of the Corollary (more precisely, a variation of the Corollary) to be the sequence of all 1's.

Example 3: Let x be the sequence of all 1's. Let $\pi_{m,n}$ be the (encoding of the) following Markov algorithm:

1. $\alpha \rightarrow 1^p$
2. $0\sigma \rightarrow \sigma 1$
3. $\beta\sigma \rightarrow \beta$
4. $1\sigma \rightarrow \lambda 0$
5. $0\lambda \rightarrow \lambda 0$
6. $1\lambda \rightarrow \lambda 1$
7. $\beta\lambda \rightarrow \lambda\beta$
8. $1\delta\lambda \rightarrow \delta 1\rho$
9. $\rho 1 \rightarrow 1\rho$
10. $\rho 0 \rightarrow 0\rho$
11. $\rho\beta \rightarrow \beta\rho$
12. $\rho * \rightarrow \sigma *$
13. $\beta 1 \rightarrow \beta$
14. $\beta * \rightarrow *$
15. $1 * \rightarrow *1$
16. $\delta * \rightarrow \delta$
17. $1\delta \rightarrow \delta$
18. $\delta \rightarrow .$
19. $\gamma 1 \rightarrow 1\gamma$
20. $\gamma 0 \rightarrow 0\gamma$
21. $\delta \rightarrow \sigma *$
22. $\rightarrow \alpha^q 1^r \delta\beta\gamma$

where $p = [\frac{n}{2^m}]$,
$q = 2^m$, and
$r = n-p\cdot q$
note: $r \leq 2^m$

$\pi_{m,n}$ computes x^n by printing out n 1's in 2^m groups of $[\frac{n}{2^m}]$ 1's. $\pi_{m,n}$ also computes x^i, given i, by computing x^n and then marking off i bits of x^n. Now for some constant c_m, $[\frac{n}{2^m}] \leq |\pi_{m,n}| \leq [\frac{n}{2^m}]+c_m$. Let Φ be the measure which counts the total number of applications of formulae which increase the number of (non-blank) symbols on the tape during a computation. For $\pi_{m,n}$ we count only the applications of formulae 1, 21, 22. Clearly, $\Phi_{\langle \pi_{m,n} \rangle}(n) = 2^m+2$. Letting $t_i(n) = 2^i+2$ and $f_i(n)=|\pi_{i,n}|$ it follows that (1) $\forall i \forall n.\ {}^*K_\Phi^{t_i}(x^n;n) \leq f_i(n)$ and (2) $\forall i \forall n.\ {}^*K_\Phi^{t_i}(x^n;n) > f_{i+1}(n)$, i.e. $\forall i.\ x \in {}^*C[f_i|t_i]$ and $\forall i.\ x \notin {}^*C[f_{i+1}|t_i]$. This merely illuminates the simple idea that if we wish to print out n 1's in fixed groups of 1's then as we decrease the number of 1's that we print out in a group, we must increase the number of groups that we print out.

This is a slightly different statement than that in the Corollary. A crucial point in establishing this was that $\pi_{m,n}$ computes x^i, given i, for $i \leq n$ by first computing x^n and then taking i bits of x^n. In order to establish this result for the complexity classes $C_\Phi[f|t]$ we must for each $i \leq n$ compute x^i, given i, within $t(i)$ steps rather than $t(n)$ steps. It is not clear that there exist programs similar to $\pi_{m,n}$ which do this and still allow one to prove that a trade-off exists.

We remark finally that by a similar argument one can show, letting $f_i(n) = 2^i+c$ and $t_i(n) = [\frac{n}{2^i}]$, for the Φ as above and the sequence x of all 1s that (1) $\forall i.\ x \notin {}^*C_\Phi[f_i|t_i]$ and (2) $\forall i.\ x \in {}^*C_\Phi[f_i|t_{i+1}]$.

References

1. Barzdin, J. "Complexity of Programs to Determine Whether Natural Numbers Not Greater Than n Belong to a Recursively Enumerable Set," Soviet Math Doleladi, Vol. 9, (1968) No. 5, pp. 1251-1254.

2. Blum, M. "A Machine Independent Theory of the Complexity of Recursive Functions," JACM, Vol. 14, (1967) pp. 322-336.

3. Chaitin, G. "On the Length of Programs for Computing Finite Binary Sequences," JACM, Vol. 13, (1966) No. 4, pp. 547-569.

4. Chaitin, G. "On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations," JACM, Vol. 16, (1969) No. 1, pp. 145-159.

5. Daley, R. "Pseudo-Recursiveness and Pseudo-Randomness Within Minimal-Program Complexity Hierarchies," Ph.D. Dissertation, Carnegie-Mellon University, 1971.

6. Kolmogorov, A. "Three Approaches for Defining the Concept of Information Quantity," Information Transmission, Vol. 1, (1965) pp. 3-11; also Selected Translations in Math. Stat. and Prob. Vol. 7 (1968), AMS Publications.

7. Kolmogorov, A. "Logical Basis for Information and Probability Theory," IEEE Transaction in Info. Theory, Vol. IT-14, (1968) pp. 662-664.

8. Loveland, D. "A Variant of the Kolmogorov Concept of Complexity," Information and Control, Vol. 15, (1969) pp. 510-526.

9. Loveland, D. "Minimal-Program Complexity Measures," Proceedings ACM Symp. on Theory of Computing, (1969) pp. 61-65.

10. Markov, A. "Theory of Algorithms," (English Translation), National Science Foundation, (1961) Washington, D. C.