SIZE ARGUMENTS IN THE STUDY OF COMPUTATION SPEEDS

J. Hartmanis

Technical Report

No. 70-73

August 1970

Department of Computer Science
Upson Hall
Cornell University
Ithaca, New York 14850

SIZE ARGUMENTS IN THE STUDY OF COMPUTATION SPEEDS*

J. Hartmanis

ABSTRACT.

In this paper we use arguments about the size of the computed functions to investigate the computation speed of Turing machines. It turns out that the size arguments yield several new results which we have not been able to obtain previously by diagonalization. For example, we show that for arbitrarily complex running times $T(n)$ there exist functions which can be computed on a one-tape Turing machine in time $T(n)$ but not in time $t(n)$ provided

$$\lim_{n\to\infty} \frac{t(n)}{T(n)} = 0 .$$

The same result is also shown to hold for many-tape machines. We show furthermore, that there exist arbitrarily complex computations for which one-tape machines are slower than two-tape machines by a factor equal to the logarithm of the computation time of the two-tape machine. We conclude by discussing several other computational complexity measures and compare results obtained by diagonalization and size arguments.

---

* This research has been supported in part by National Science Foundation Grant No. GJ-155.

I. Introduction.

In the study of computational complexity we have relied heavily on diagonalization arguments to exhibit functions with well-defined computation speeds. In general, these results prove for a given computational complexity measure the existence of a recursive function g, such that for arbitrarily large computation times $T(n)$ there exist functions computable in time $g[T(n)]$ but not in time $T(n)$. Thus the function g tells us how well we can bound the computation speed for difficult computations in a given measure. For the computation speeds of one-tape as well as many-tape Turing machines we have only been able to show by means of diagonal arguments that $g[T(n)]$ does not have to be much larger than $T(n) \log T(n)$ [1,2]. For one-tape Turing machines the result has the following form [1]: for arbitrarily large running-times $T(n)$ there exist functions which can be computed on a one-tape Turing machine in time $T(n) \log T(n)$ but not in time $t(n)$, provided

$$\lim_{n\to\infty} \frac{t(n)}{T(n)} = 0 .$$

For many tape machines the corresponding result asserts [2]: if $T(n)$ is a running-time then there exists a function which can be computed on a multi-tape Turing machine in time $T(n)$ but not in time $t(n)$, provided

$$\lim_{n\to\infty} \frac{t(n) \log t(n)}{T(n)} = 0 .$$

It should be recalled that for small running times, $T(n) \leq n^2$, considerably stronger results have been obtained for one-tape machines by means of "information transfer" arguments [1,3,4]. Unfortunately, these arguments could not be extended directly to arbitrarily large running times.

In this paper we use arguments about the size of the computed functions to show that for arbitrarily large running times $T(n)$ of one-tape Turing machines we can exhibit functions which can be computed in time $T(n)$ but not in time $t(n)$, provided

$$\lim_{n\to\infty} \frac{t(n)}{T(n)} = 0 .$$

This result also holds for many-tape Turing machines.

By the same reasoning it can also be shown that many-tape Turing machines are faster than one-tape Turing machines for arbitrarily long computations. We show that if $T(n)$ is the running time of a one-tape Turing machine then there exists a function which can be computed in time $T(n)$ on a two-tape Turing machine but cannot be computed in time $t(n)$ on any one-tape machine provided

$$\lim_{n\to\infty} \frac{t(n)}{T(n)\ \log\ T(n)} = 0 .$$

Thus the computation time on the one-tape Turing machine cannot grow more slowly than $T(n) \log T(n)$.

To further illustrate the power of size arguments, we show that when we use a fixed universal Turing machine then for arbitrarily large time bounds there exist functions whose computation time cannot be improved by an arbitrary positive constant factor, as is the case for general Turing machine computations [1,5]. In conclusion we take a quick look at three other computational complexity measures and discuss why for two of them size argmments yield sharper results than diagonalization arguments. For the third measure, which is based on one-tape Turing machines with "jump" operations, we show that diagonalization yields for complexity classes of zero-one functions as sharp results as we can get by size arguments for complexity classes of unbounded functions.

II. Size Arguments.

In the first part of this paper we are concerned with the computation speeds of one-tape Turing machines. We say that a function $f(n)$ mapping positive integers into positive integers is computable in time $T(n)$ on a one-tape Turing machine if and only if there exists a one-tape Turing machine M which started with input n written (in binary notation) on its tape halts in $T(n)$ or fewer operations with $f(n)$ printed (in binary notation) on its tape. Symbolically we write

$$f \in C^{1}_{T(n)}$$

where $C^{1}_{T(n)}$ denotes all functions computable in time $T(n)$ on a one-tape Turing machine. Without any loss of generality we assume that our one-tape Turing machines move their head on each operation and that the tape is only unbounded to the right. The machine is started on the left most tape square which contains the first digit on the input sequence.

The same definition of computation time holds for many-tape machines with the provision that the input is written on the input tape and that the output must be printed on the output tape. We let $C_{T(n)}$ denote all functions computable in time $T(n)$ on multi-tape Turing machines and write $C^{k}_{T(n)}$ to denote the set of functions computable in time $T(n)$ on k-tape Turing machines. We will add $\{0,1\}$ to the superscript of the complexity class to show that we are considering only zero-one functions.

For a one-tape Turing machine with input n we define on each boundary between two adjacent tape squares as <u>crossing sequence</u> [1,3] the sequence of states $s_1, s_2, s_3, \ldots$ the machine is in as it crosses this boundary during the computation when started on input n. Thus s_k is the state M is in on the k-th crossing of this boundary. Note that if M halts on input n then only a finite number of its crossing sequences are non-null and the sum of the lengths of all crossing sequences gives the number of operations performed by the Turing machine on input n before halting.

Our first result simply points out that any one-tape Turing machine which halts must generate different crossing sequences on all boundaries between tape squares on the initially blank part of the tape. For a detailed discussion of crossing sequence arguments see [3,1].

<u>Lemma</u>: If a one-tape Turing machine M halts on input n then all non-null crossing sequences which M generates on its initially blank part of the tape must be distinct.

<u>Proof</u>: If the machine M ever generates two identical non-null crossing sequences on the initially blank part of the tape then it must generate an unbounded number of such crossing sequences. To see this just observe that all the information carried across a boundary by M is contained in the crossing

sequence. On the initially blank part of the tape there cannot be any information on the tape as the machine generates the original duplication of the crossing sequence. Thus all the information carried across the first identical crossing sequence C_1 which results in generation of C_2, $C_2 = C_1$, is also carried across C_2 and an identical crossing sequence C_3, etc. will be generated further to the right on the tape. Thus M generates an unbounded number of crossing sequences and does not halt, as was to be shown.

Lemma: Let M be a one-tape Turing machine with q, $q \geq 2$, states which for an input of length ℓ writes on $L(\ell)$ tape squares $L(\ell) > \ell$. Then M performs at least

$$\frac{1}{q}[L(\ell) - \ell][\log_q [L(\ell) - \ell] - 1]$$

operations for this input before halting.

Proof: By the previous lemma M must generate at least $L(\ell) - \ell$ different crossing sequences before halting. Since M has q states it can generate q^k different crossing sequences of length k. If M does the most economical thing and generates the shortest possible crossing sequences, then it has to generate all crossing sequences up to length $d - 1$ such that

$$\sum_{k=1}^{d} q^k \geq L(\ell) - \ell .$$

Note that any other pattern of generating the $L(\ell) - \ell$ different crossing sequences requires more operations. Thus

$$\frac{q^{d+1}-1}{q-1} \geq L(\ell) - \ell$$

and for $q \geq 2$,

$$q^{d+1} \geq L(\ell) - \ell$$

or

$$d + 1 \geq \log_q \, [L(\ell) - \ell] \; .$$

Since M does not necessarily have to generate all crossing sequences of length d we now compute how many operations are needed to generate all crossing sequences up to and including length $d - 1$. Since there are q^k crossing sequences of length k the number of operations required is not less than

$$\sum_{k=1}^{\log_q [L(\ell)-\ell]-2} kq^k \geq [\log_q \, [L(\ell) - \ell] - 1] \, q^{\log_q [L(\ell)-\ell]-1}$$
$$\geq \frac{1}{q} \, [L(\ell) - \ell](\log_q \, [L(\ell) - \ell] - 1) \, ,$$

as was to be shown.

Note that for $L(\ell)$ which grow faster than linearly we can replace this formula by

$$\frac{1}{q}(L(\ell)-\ell)(\log_q[L(\ell)-\ell]-1) \geq \frac{1}{2q} \, L(\ell)\log_q L(\ell) = \frac{\log_2 q}{2q} \, L(\ell)\log_2 L(\ell) \; ,$$

for sufficiently large values of $L(\ell)$. From the previous lemma and observation we get the next result.

<u>Theorem</u>: Let M be a q state, one-tape Turing machine which computes the function $f(n) \geq 2^n$. Then for large values of n the computation time $T(n)$ of $f(n)$ must satisfy the inequality

$$T(n) \geq \frac{1}{2q} (\log_2 f(n)) \cdot (\log_q \log_2 f(n)) .$$

The above result relates the computation speed to the size of the function computed. Next we show that we can construct functions whose computation time is very nearly the predicated minimal time and thus show that their computation time cannot be improved by more than a constant factor.

<u>Theorem</u>: There exist arbitrarily large functions $f_i(n)$ such that $f_i(n)$ is computable in $T_i(n)$ operations on a one-tape Turing machine but not in $t(n)$ operations provided

$$\lim_{n\to\infty} \frac{t(n)}{T_i(n)} = 0 .$$

In other words, there exist arbitrarily large $T_i(n)$ such that

$$\lim_{n\to\infty} \frac{t(n)}{T_i(n)} = 0$$

implies that

$$C_t^1 \subsetneq C_{T_i}^1$$

<u>Proof</u>: We first observe that there exist arbitrarily large running times $T'(n)$ in which a Turing machine can mark off at least $\log_2 T'(n)$ tape squares for large n. As a matter of fact for any large running time $T''(n)$ achieved by a two-symbol Turing machine the machine must use more than $\log_3 T''(n)$ tape squares. Since otherwise it must repeat some total state, tape head and tape pattern configuration and therefore cannot halt. These Turing machines can easily be modified to yield at least $\log_2 T'(n)$ marked tape squares in $T'(n)$ operations.

Furthermore, we observe that there exists a one-tape Turing machine which for any input of length ℓ can mark-off 2^ℓ tape squares in less than $c \circ 2^\ell \ell$ operations. This can be done by just counting on the ℓ tape squares from 1 to 2^ℓ and after each counting move the ℓ tape squares over to the right by one tape square.

We now consider the Turing machine which first marks off

$$g(n) \geq \log_2 T'(n)$$

tape squares in $T'(n)$ operations and then marks off $2^{g(n)}$ tape squares by writing a one followed by $2^{g(n)}$ zeros in $c2^{g(n)}g(n)$ operations. Thus this machine computes

$$f(n) = 2^{2^{g(n)}}$$

in

$$T(n) \leq c \circ 2^{g(n)} g(n) + T'(n) \leq c_1 2^{g(n)} g(n)$$

operations. On the other hand, our lemma asserts that any one-tape, q state Turing machine which computes

$$f(n) = 2^{2^{g(n)}}$$

must have a running time $t(n)$ satisfying

$$t(n) \geq \frac{\log_q 2}{2q} \circ g(n)\ 2^{g(n)}$$

for large n. This implies that $f(n)$ is computable in

$$T(n) \leq c_1 g(n)\ 2^{g(n)}$$

but not computable in $t(n)$ such that

$$\lim_{n\to\infty} \frac{t(n)}{T(n)} = 0 ,$$

as was to be shown.

It would be very interesting to find out whether the same result holds when we restrict ourselves to zero-one functions. So far we have not been able to make any improvement for zero-one functions of the Hennie-Stearns result mentioned in the

Introduction [2]. Nevertheless, the above result shows that for arbitrarily large functions we can bound the computation times very tightly and obtain a much stronger result.

We now use the above result to show that for arbitrarily long computations two-tape machines are faster than one-tape machines.

<u>Theorem</u>: Let $T(n)$ be the running-time of a one-tape Turing machine. Then there exists a function $f(n)$ which is computable in time $T(n)$ on a two-tape machine and the running time $t(n)$ of any one-tape Turing machine with q states computing $f(n)$ must satisfy the inequality

$$t(n) \geq \frac{1}{4q} \circ T(n) \circ \log_q T(n) ,$$

for large n .

<u>Proof</u>: Let the two-tape machine M operate on the input tape as a one-tape machine whose running time is $T(n)$ operates on its tape. On the output tape M prints a sequence of ones. Thus M computes

$$f(n) = 2^{T(n)} - 1$$

in time $T(n)$.

On the other hand, by a previous lemma we know that a q state, one-tape machine requires for large n time

$$t(n) \geq \frac{1}{2q} (\log_2 f(n))(\log_q \log_2 f(n))$$

to compute $f(n)$. From this it follows that

$$t(n) \geq \frac{1}{4q} T(n) \cdot \log_q T(n)$$

for large n . This completes the proof.

We can express this result in a different way:

<u>Corollary</u>: Let $T(n)$ be the running time of a one-tape Turing machine. Then there exists a function $f(n)$ computable in $T(n)$ operations on a two-tape machine and not computable in time $t(n)$ on a one-tape machine provided

$$\lim_{n\to\infty} \frac{t(n)}{T(n)\log_2 T(n)} = 0 .$$

It would be very interesting to prove the same result without the use of the size argument. Unfortunately, we have no such proof and thus it is still an open problem whether for the computation of zero-one functions, two-tape machines are faster than one-tape machines for arbitrarily long computations. Recall that up till now it was only known that for real-time computations two-tape machines are faster than one-tape machines [2,4].

Furthermore, even using size arguments we have not been able to show for $k \geq 3$ that k-tape machines are faster than (k-1)-tape machines for arbitrarily long computations.

The next result shows that size arguments can very simply yield new results for time bounds of many-tape Turing machine computations.

<u>Theorem</u>: Let $T(n)$ be the running time of a multitape Turing machine. Then there exists a function $f(n)$ which can be computed in time $T(n)$ but cannot be computed in time $t(n)$ by any multitape Turing machine provided

$$\lim_{n\to\infty} \frac{t(n)}{T(n)} = 0 .$$

In other words, for running time $T(n)$

$$\lim_{n\to\infty} \frac{t(n)}{T(n)}$$

implies that

$$C_t \subsetneq C_T .$$

<u>Proof</u>: Let $T(n)$ be the running time of a k-tape machine M . Construct a (k+1)-tape machine M' so that the first k tapes of M' are used as tapes of the machine M . The $k + 1$ tape is used as output tape and M' prints a sequence of ones, one on each operation. Thus M' computes

$$f(n) = 2^{T(n)} - 1 .$$

Any other multitape Turing machine running in time $t(n)$ can compute at best a function of size

$$g(n) \leq q^{t(n)}$$

If

$$\lim_{n\to\infty} \frac{t(n)}{T(n)} = 0$$

then, for large n,

$$q^{t(n)} < 2^{T(n)} - 1$$

and we see that

$$C_t \subsetneq C_T \; .$$

as was to be shown.

Again the open problem is to determine whether this result holds for complexity classes of zero-one functions. Recall that so far the best diagonalization result which holds for zero-one functions can only assert that

$$\lim_{n\to\infty} \frac{t(n)\log t(n)}{T(n)} = 0$$

implies that $C_t \subsetneq C_T$ [2].

III. Complexity Classes Defined by Universal Turing Machines.

We know that for larger computation times any many-tape or one-tape Turing machine computation can be speeded up by any positive factor by selecting a larger Turing machine with more tape symbols [1,5]. In this section we show that this is not the case when we use a fixed universal Turing machine as our computer.

Let M be a one-tape, universal Turing machine which interprets the first binary string up to the marker $\#$ on its tape as an algorithm (description of another one-tape Turing machine) and the second binary string as the input to which the algorithm is to be applied. The output is to be placed on the tape to the right of the input between appropriate markers.

Clearly this universal Turing machine with its algorithms defines a computational complexity measure if we assign to each algorithm A_i the step-counting function $T_i(n)$ given by the number of operations performed by M with algorithm A_i on input n before halting. We let $C^M_{T(n)}$ denote the set of all functions computable almost everywhere in the bound $T(n)$.

Theorem: There exists a universal Turing machine M such that for any recursive $t(n)$ there exists a running time $T(n) \geq t(n)$ and a constant k such that

$$C^M_{T(n)/k} \subsetneq C^M_{T(n)} .$$

Proof: By a previous lemma we know that a q-state, one-tape machine requires $T_i(n)$ operations to compute $f_i(n)$, with

$$T_i(n) \geq \frac{\log_q 2}{2q} (\log_2 f_i(n))\log_2 \log_2 f_i(n))$$

for large n. At the same time we can construct a universal Turing machine which can compute arbitrarily complex functions $f_i(n)$ in time

$$T_i(n) \leq c_i \circ (\log_2 f_i))(\log_2 \log_2 f_i(n)) ,$$

where the constant depends on the size of the algorithm for $f_i(n)$. This is achieved by first recalling (from one of our previous proofs) that there exist single-tape Turing machines which compute arbitrarily large functions within a constant times of the minimal time mentioned above. Secondly, observing that there exist universal Turing machines which for each algorithm can simulate the computation by not taking more than a constant times the number of operations performed by the original machine (the constant depends on the algorithm). This simulation is performed by always moving along a copy of the algorithm which is being executed and thus insuring that for each simulated operation no more than a fixed number of operations are required.

Combining the two inequalities we see that for

$$k > \frac{c_i 2q}{\log_q 2}$$

we cannot speed up the computation of $f_i(n)$ by a factor k and thus conclude that

$$C^M_{T_i/k} \subsetneq C^M_{T_i}$$

as was to be shown.

Again the fascinating open problem is to determine whether the same result holds for complexity classes of zero-one functions. It would also be interesting to show that this result holds for all universal Turing machines. The best result we have been able to obtain for this computational complexity measure for zero-one function is stated below.

Theorem: There exists a universal Turing machine M and arbitrarily large running times $T_i(n)$ such that

$$\lim_{n\to\infty} \frac{t(n)}{T_i(n)} = 0$$

implies that

$$C^{M\{0,1\}}_{t(n)} \subsetneq C^{M\{0,1\}}_{T(n)\log T(n)}$$

Proof: The proof is obtained by an obvious modification of the proof for the corresponding result about one-tape machines in [1].

IV. Other Turing Machine Variants.

To further illustrate the power of the size argument and its relation to diagonalization, we discuss three other computational complexity measures based on modified one-tape Turing machines.

It should be recalled that our previous diagonalization arguments to prove that some function cannot be computed in time $t(n)$ have involved two different computations [1,2,5]. One computation just simulated all other Turing machines and changed the result to construct the desired function (diagonalized); the other computation kept track of how many simulation operations had been performed and shut the computation off when it tried to exceed a given bound. For one-tape Turing machines the simulation part is quite easy but the running of two processes on one tape is more difficult and this is reflected in the rather weak result we can get by means of simulation [1]. For many tape Turing machines the problem is reversed [2,5]: running two processes in parallel is easy since we just add the necessary tapes and expand the machine to behave like the two desired machines. On the other hand, simulation is more difficult since we have to simulate machines with arbitrarily many tapes on a machine with a fixed number of tapes. Thus in this case the simulation causes the weakness in the corresponding result [2].

To gain further insight in these problems, we now define three new complexity measures. The first two models overcome the problem of running two processes simultaneously on a one-tape machine. Unfortunately, though they solve the paralelism problem, they complicate the simulation problem and we have not been able to exploit the additional power to prove as good results for zero-one functions as we can obtain by size arguments for unbounded functions.

The first computational complexity measure is defined by using one-tape Turing machines and counting the number of symbols printed, rather than all operations performed. This "ink" based computational complexity measure has the advantage that a search for a symbol or moving the head from one computation to another does not cost anything. At the same time we have not been able to devise a way where we can simulate other Turing machines without using ink if they do not use ink. Thus it is still an open problem whether for arbitrarily large ink bounds $I_i(n)$

$$\lim_{n\to\infty} \frac{I(n)}{I_i(n)} = 0$$

implies that there exists a zero-one function which can be computed with $I_i(n)$ ink but not with $I(n)$ ink. On the other hand, by using size arguments we can easily show that such a result holds for unbounded functions.

A similar situation arises when we consider computational complexity measures based on the number of operations performed by one-tape Turing machines with the ability to jump to the nearest specified symbol in one operation. These machines operate as regular Turing machines until they enter a special state which transfers their heads in one operation to a nearest tape symbol specified by the state (if no such symbol is printed on the tape, the machine halts). Again, these jump operations make the running of two simultaneous processes easy but the simulation becomes more difficult. The difficulty is very similar to the one arising for many tape Turing machines [2,5]; that is, we must simulate machines with arbitrarily many symbols to which they can jump on a machine with a fixed number of jump-symbols. Thus again, just as in the previous case, we have not been able to obtain as sharp a result for zero-one functions as we can obtain by size arguments for unbounded functions.

Finally, we discuss a complexity measure for which we can obtain the same result for complexity classes of zero-one functions by means of diagonalization as for complexity classes of unbounded functions with size arguments.

Consider one-tape Turing machines with special marker, $\#$, and a "jump" state which can be used to transfer the head in one operation to the nearest specified sub-sequence on the tape as follows: let x be any binary sequence placed between two markers, $\# x \#$, then by placing its head on the left marker and entering the jump state the head is transferred in one operation to the nearest left marker bounding the sequence, $\# x \#$, if such a second sequence exists, otherwise the machine halts. Again we obtain a computational complexity measure by counting the number of operations performed by such a Turing machine with jump instructions. Fortunately, for this measure we get the following result.

<u>Theorem</u>: Let $T(n) \geq n$ be the running time of a Turing machine. Then there exists a zero-one function computable on a Turing machine with jump instructions in time $T(n)$ but not in time $t(n)$ provided

$$\lim_{n\to\infty} \frac{t(n)}{T(n)} = 0 \; ;$$

that is

$$C_t^{J\{0,1\}} \subsetneq C_T^{J\{0,1\}} \; ,$$

where J indicates that we are considering T.M.'s with jump operations.

<u>Proof</u>: The proof is obtained by a standard diagonalization argument used in [1]. The only trick is that we reserve the shortest sequences $\# 0 \#$, $\# 1 \#$ as markers to jump back and forth between the simulation and the computation counting up to $T(n)$. The longer sequences of symbols are used to encode the simulated computation. Since these encodings and jumping between the two computations can only increase the computation time by a constant factor, the result follows easily.

V. Conclusion.

In this paper we showed that size arguments can give very sharp results about complexity classes and solved some problems which we have not been able to resolve by means of diagonalization. At the same time we had to state these results for complexity classes consisting of all functions computable in a given bound, whereas diagonalization usually yields results about zero-one functions. This has left us with several tantalizing conjectures for complexity classes of zero-one functions which we know to be true for unrestricted complexity classes. It would be rather surprising if the results for complexity classes of zero-one functions would turn out to be different than those obtained in this paper for unrestricted complexity classes.

VI. References.

1. Hartmanis, J. "Computational Complexity of One-Tape Turing Machine Computations," JACM 15 (1968), 325-339.

2. Hennie, F.C., and R.E. Stearns. "Two-Tape Simulation of Multi-Tape Turing Machines," JACM 13 (1966), 533-546.

3. Hennie, F.C. "One-Tape, Off-Line Turing Machine Computations," Information and Control 8 (1965), 553-578.

4. Rabin, M.O. "Real Time Computation," Israel Journal of Mathematics 1 (1964), 203-211.

5. Hartmanis, J., and R.E. Stearns. "On the Computational Complexity of Algorithms," Trans. of the American Math. Soc. 117 (1965), 285-306.