ON TIME VERSUS SPACE[+]

John Hopcroft[++]

Wolfgang Paul[+++]

Leslie Valiant[++++]

TR 75-264

December 1975

Computer Science Dept.
Cornell University
Ithaca, NY 14853

[+]This research was supported in part by DAAD (German Academic Exchange Service) Grant No. 430/402/563/5 and the Office of Naval Research under Grant N-00014-67-A-0077-0021.

[++]Author's address: Cornell University, Ithaca, NY 14853.

[+++]Author's address: Cornell University, Ithaca, NY 14853 and Universität des Saarlandes, Saarbrücken, Germany.

[++++]Author's address: University of Leeds, Great Britain.

Abstract:

It is shown that every deterministic multitape Turing machine of time complexity t(n) can be simulated by a deterministic Turing machine of tape complexity t(n)/log t(n). Consequently for tape constructable t(n), the class of languages recognizable by multitape Turing machines of time complexity t(n) is strictly contained in the class of languages recognized by Turing machines of tape complexity t(n). In particular the context sensitive languages can not be recognized in linear time by deterministic multitape Turing machines.

Keywords and phrases: Turing machines, time complexity, tape complexity.

CR Categories: 5.25

1. Introduction

The main result of this paper is to settle in the affirmative the fundamental question as to whether space is strictly more powerful than time as a resource for deterministic multi-tape Turing machines.

We establish for all functions $t(n)$ that the class of sets accepted in time $t(n)$ is contained within the class of sets accepted in space $t(n)/\log t(n)$. This implies by the standard diagonalization arguments [11] that for functions $t(n)$ and $\delta(n)$ which are both constructable on tape $t(n)$ the class of sets accepted in time $t(n)$ is properly contained in the class of sets accepted in space $\delta(n)t(n)/\log t(n)$ provided $\inf \delta(n) \to \infty$.

One consequence of the above result is that the context-sensitive languages can not be recognized in linear time on a deterministic multi-tape Turing machine. In fact there exists a deterministic context-sensitive language which cannot be recognized in time $n\log n/A^{-1}(n)$ on a deterministic multi-tape Turing machine where $A^{-1}(n)$ is the inverse of Ackermann's function.

2. Efficient space simulation of time bounded Turing machines.

Let DTIME(t(n)) [NTIME(t(n))] be the class of sets accepted by deterministic [nondeterministic] multi-tape Turing machines of time complexity $t(n)$. Let DSPACE(s(n)) [NSPACE(s(n))] be the class of sets accepted by deterministic [nondeterministic] multi-tape

Turing machines of space complexity s(n).

We present an algorithm for simulating a deterministic multi-tape Turing machine of time complexity t by a deterministic Turing machine of space complexity t/logt. For ease in understanding the construction involved we first present a nondeterministic simulation.

Partition each tape into blocks of size $t^{2/3}$. Next modify the time bounded Turing machine so that it is block respecting that is, tape heads will cross boundaries between blocks only at times which are integer multiples of $t^{2/3}$. The modified Turing machine must be so constructed so that it runs slower by at most a constant factor c. To do this block boundaries are marked on the tapes. An additional tape is added with one block marked on it. The head on this new tape simply moves back and forth over the block and serves as a clock to indicate multiples of $t^{2/3}$ units of time. If either head on one of the original tapes attempts to cross a block boundary at a time other than a multiple of $t^{2/3}$, the simulation temporarily halts until the clock tape indicates the next multiple.

A difficulty arises in that a tape head may simply move back and forth between two adjacent tape cells which are in different blocks. In this case the simulation would be slower by a factor of $t^{2/3}$. This difficulty is overcome as follows.

Let

w_1	w_2	...	w_r

be an inscription of the jth tape of the original Turing machine after $mt^{2/3}$ steps where $|w_i| = t^{2/3}$. Then the corresponding inscription for the block respecting Turing machine after $cmt^{2/3}$ steps is

	w_1^r	w_2^r		w_{r-1}^r
w_1	w_2	w_3	...	w_r
w_2^r	w_3^r			

where w^r stands for w reversed. The extra tracks are used to guarantee that $t^{2/3}$ moves can be simulated without crossing a block boundary. After $t^{2/3}$ such moves of actual simulation, the next $(c-1)t^{2/3}$ moves are used to update the tapes of the block respecting machine. The details of the simulation and of marking the block boundaries are left to the reader. (For help, consult the proof of Theorem 10.3 in [7]).

Without loss of generality let M be a block respecting deterministic k-tape Turing machine of time complexity t and let w be an input for M. Divide the computation of M on input w into time segments. Each segment Δ corresponds to a time period of length $t^{2/3}$. Note that a tape head can cross a boundary only at

the end of a segment. Since the original Turing machine makes at most t moves and there are always $t^{2/3}$ moves between crossings of block boundaries, there are at most $t^{1/3}$ time segments Δ.

Let $h(i,\Delta)$ be the position of the tape head on the ith tape of M after time segment Δ. Let $h(\Delta) = [h(1,\Delta),\ldots,h(k,\Delta)]$ and let $h = [h(1),\ldots,h(t^{1/3})]$.

From the sequence of head positions h we construct a directed graph G with a vertex corresponding to each time segment of the computation. Let $v(\Delta)$ be the vertex corresponding to time segment Δ. For each tape i, let Δ_i be the last time segment prior to Δ such that the ith head was scanning the same block during the segment Δ_i as during the segment Δ. The edges of G are $v(\Delta-1) \rightarrow v(\Delta)$ and for $1 \leq i \leq k$: $v(\Delta_i) \rightarrow v(\Delta)$. Because there are only $t^{1/3}$ time segments Δ, the above graph has at most $t^{1/3}$ vertices. To write down a description of the graph requires space on the order of $(k+1)\ t^{1/3} \cdot \log t$ since approximately logt bits are needed to specify each edge.

Let $c(i,\Delta)$ be the content after time segment Δ of that block on the ith tape of M, scanned by the tape head during the time segment Δ. Let $c(\Delta) = [c(1,\Delta),\ldots,c(k,\Delta)]$. Let $f(\Delta)$ be the initial contents of those blocks, which are visited by M for the first time during the time segment Δ. Now with each vertex $v(\Delta)$ in G we can associate the information $c(\Delta)$ and $f(\Delta)$.

Let $q(\Delta)$ be the state of M after time segment Δ and let

$q = [q(1),\ldots,q(t^{1/3})]$. In order to determine the outcome of the original computation of M, we need only determine the state $q(\Delta)$ after the final time segment Δ.

Suppose we have guessed the sequence of head positions h and the sequence of states q. We want then to simulate M without using too much space and to verify that we guessed h and q correctly. Because M is deterministic and block respecting, the following holds for any Δ:

(2.1): $q(\Delta)$, $h(\Delta)$ and $c(\Delta)$ can be uniquely determined from $q(\Delta-1)$, $h(\Delta-1)$, $c(\Delta_1),\ldots,c(\Delta_k)$ and $f(\Delta)$ by direct simulation of time segment Δ. The simulation requires space to store the contents of k blocks or $O(kt^{2/3})$.

(2.2): To store $c(\Delta)$ requires space $O(kt^{2/3})$.

$f(\Delta)$, $q(\Delta-1)$ and $h(\Delta-1)$ can be determined from the guessed sequences q and h, and the edges into vertex $v(\Delta)$ in the graph G give us the vertices associated with $c(\Delta_1),\ldots,c(\Delta_k)$. This suggests several strategies to simulate M. Our strategy will be first to determine c(1), then c(2), c(3) and so on.

Now observe, that in order to carry out the whole simulation we need not keep in memory the contents of the blocks corresponding to all vertices since we can go back and reconstruct certain blocks whenever we need them. The question is what is the minimum number of blocks one must store at any one time in order to carry out the simulation. To answer this question we study a game on graphs.

Let G_k be the set of all finite directed acyclic graphs with indegree at most k. Vertices with indegree 0 are called input vertices. The game consists in placing pebbles on the vertices of such a graph G according to the following rules:

(1) A pebble can alway be placed on an input vertex.

(2) If all fathers of a vertex v have pebbles, then a pebble can be placed on vertex v.

(3) A pebble can be removed at any time.

The goal of the game is to eventually place a pebble on a particular vertex v, designated in advance, by a scheme which minimizes the maximum number of pebbles simultaneously on the graph at any instance of time. Let $P_k(n)$ be the maximum over all graphs in G_k with n vertices of the number of pebbles required to place a pebble on an arbitrary vertex of such a graph. We will show that for each k, $P_k(n)$ is $0(n/\log n)$.

Lemma 1: For each k, $P_k(n)$ is $0(n/\log n)$.

Proof: For convenience let $R_k(n)$ be the minimum number of edges of any graph in G_k which requires n pebbles. Showing that $R_k(n) \geq cn\log n$ for some c is equivalent to proving that $P_k(n)$ is $0(n/\log n)$.

Let $G = (V,E)$ be a graph in G_k with $R_k(n)$ edges which requires n pebbles. Let V_1 be the set of vertices of G to which a pebble can be moved using n/2 or fewer pebbles. Let

$V_2 = V - V_1$,

$E_1 = \{(u \to v)/(u \to v) \in E, u,v \in V_1\}$,

$E_2 = \{(u \to v)/(u \to v) \in E, u,v \in V_2\}$,

$G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$.

Let $A = E - (E_1 \cup E_2)$, that is A is the set of edges from vertices in V_1 to vertices in V_2 .

We claim that there exists a vertex in G_2 which requires $n/2-k$ pebbles if the game is played on G_2 only. Otherwise move a pebble to any vertex v of G with less than n pebbles by the following strategy. If v is in V_1 then only $n/2$ pebbles are needed. Thus assume v is in V_2 . Move a pebble to v in G by using the strategy for G_2. Whenever we need to place a pebble on a vertex w of G_2 which in G has a father in V_1, move pebbles one at a time to each father of w in V_1. Since w has at most k fathers in V_1 at most $n/2+k$ pebbles are ever placed on vertices in V_1. As soon as a pebble is placed on w remove all pebbles from vertices in V_1. Hence at most $n/2 - k-1$ pebbles are ever used, a contradiction. Thus G_2 must have at least $R_k(n/2-k)$ edges.

Next observe that G_1 has a vertex which requires at least $n/2-k$ pebbles. This follows from the fact that a vertex requiring n pebbles must have an ancestor which requires at least $n-k$ pebbles. Thus G_1 must have at least $R_k(n/2-k)$ edges.

Now either $|A| \geq n/4$ in which case $R_k(n) \geq 2R_k(n/2-k) + n/4$ or else $|A| < n/4$. In the latter case, pebbles can be placed simultaneously on all vertices of V_1 which are tails of edges in A using at most $3n/4$ pebbles in the process. Leaving $n/4$ pebbles on these vertices we have $3n/4$ pebbles free after this has been accomplished. Thus G_2 must require $3n/4$ pebbles for otherwise G would not require n pebbles. Now a graph which requires $3n/4$ pebbles must have at least $\frac{1}{k}\, n/4$ edges more than a graph requiring $n/2$ pebbles. (This is also a consequence of the fact that a vertex requiring n pebbles must have a father requiring at least n-k pebbles).

Thus in both cases

$$R_k(n) \geq 2R_k(n/2-k) + \frac{1}{k}\, n/4 \ .$$

Solving this recurrence gives $R_k(n) \geq c\ n\log n$ for some constant c. □

Cook [2] has shown that $P_2(n) \geq c\sqrt{n}$ for some constant $c > 0$. It is an interesting question whether or not one can prove a lower bound of n/logn on the number of pebbles $P_k(n)$ for some fixed k.

Lemma 2: $\mathrm{DTIME}(t) \subseteq \mathrm{NSPACE}(t/\log t)$.

Proof: Let M be a tlogt time bounded deterministic k-tape Turing machine. We construct a nondeterministic machine M' which simulates M in space t.

Make M block respecting and guess a sequence of states q' and a sequence of head positions h'. We denote these by q' and h'

as opposed to the correct sequences q and h. Each such sequence has length at most $t^{1/3}$. It requires space $t^{1/3}$ to write down q' and space $t^{1/3}\log t$ to write down h'.

From h' construct a graph G as described earlier. G has $t^{1/3}$ vertices and requires space $t^{1/3}\log t$ to write down.

By Lemma 1 there is a strategy to move a pebble to the output node of G never using more than $t^{1/3}/\log t$ pebbles. We can assume that this strategy has at most $\tau = 2^{t^{1/3}}$ moves because there are only τ patterns of pebbles on G and there is no sense in repeating a pattern in a strategy. Having guessed the sequences q' and h', M' simulates M as follows:

```
begin
    for x = step 1 until τ do
    begin
        nondeterministically guess x^th move of above strategy;
        if x^th move places pebble on v(Δ) then
        begin
            compute and store q(Δ), h(Δ) and c(Δ);
            if q(Δ) ≠ q'(Δ) or h(Δ) ≠ h'(Δ) then reject;
            if space used ≥ 0(t) then reject;
        end else if x^th move removes pebble from v(Δ) then
            erase q(Δ), h(Δ) and c(Δ) from the working tape;
    end
    if after stage τ simulation is not complete then reject
end
```

The essential feature of this simulation is, that after stage x, M' has computed and stored $\{c(\Delta) \mid$ vertex $v(\Delta)$ has a pebble after the x^{th} move$\}$. By (2.2) storing $c(\Delta)$ for one Δ takes space $0(t^{2/3})$. Thus if M' happens to guess a strategy which uses at most $0(t^{1/3}/\log t)$ pebbles at a time (by Lemma 1 such a strategy always exists), the simulation can indeed be carried out in space $0(t/\log t)$, in which case M' accepts iff the last component of q' is the accepting state of M.

The global correctness of the above simulation is proven by induction on x and follows from (2.1), (2.2), the construction of G and the rules of the pebble game. A small but important point is that there are the edges $v(\Delta-1) \to v(\Delta)$. This guarantees that the time segment Δ of the computation of M cannot be simulated until it has been verified that $q(1),\ldots,q(\Delta-1)$ and $h(1),\ldots,h(\Delta-1)$ had been guessed correctly. The details of the correctness proof are left to the reader.

At this point the reader should be familiar with all the important ideas. We now explain how to make the simulation deterministic. There are two nondeterministic steps in the simulation algorithm. Guessing the sequences q' and h' can be replaced by cycling through all possible such sequences.

In order to determine the next move in the strategy which moves the pebbles, first construct a nondeterministic machine which given a description of G, a pattern D of pebbles on G and a number x between 1 and $2^{t^{1/3}}$ (each of which can be written down in space $t^{1/3}\log t$ or less) prints out the <u>first</u> move in a strategy which

starting from D, moves a pebble on the output vertex of G never using more than $0(t^{1/3}\log t)$ pebbles and making at most $2^{t^{1/3}\log t - x}$ moves, provided such a strategy exists.

This machine can be constructed in a straightforward way using space $0(t^{1/3}\log t)$. Using techniques from [10] it can be made deterministic in space $0(t^{2/3}\log^2 t)$. Using this machine during the simulation as a submachine, one can always from the achieved pattern of pebbles and from x deterministically find the next move in the right strategy. Thus we have shown

Theorem 1: If t is tape constructable, then

$$\mathbf{DTIME}(t) \subseteq \mathrm{DSPACE}(t/\log t).$$

Some easy corollaries of this have been stated in the introduction. In addition one can show

Corollary 1: For all t:

$$\mathbf{DTIME}(t) \subseteq \mathrm{DSPACE}(t/\log t).$$

Proof: Instead of precomputing t, successively try simulation of the proof of Theorem 1 for $t(n) = 1,2,3,\ldots$ until one can carry out the simulation.

Corollary 2: If t(n) and $\delta(n)$ are constructable on tape t(n) and $\lim \delta(n) = \infty$, then DTIME(t(n)) is properly contained in the class of sets recognizable in time $\delta(n)t(n)\log t(n)$ on space t(n).

Proof: Instead of blocksize $t^{2/3}$ choose blocksize $t/(\log\delta)^{1/2}$. A time analysis of the proof of Theorem 1 with this modification shows that each k-tape machine of time complexity t can be simulated by a k+1-tape machine of time complexity less than δt and tape complexity $0(t/\log\log \delta)$. Now an appeal to [5] yields the desired result.

It is an interesting open problem whether NTIME(t) $\subseteq$ NSPACE(t/logt). The difficulty here is that in going back and repeating a portion of a computation, we cannot be sure that the same sequence of choices is made the second time.

3. An application of Lemma 1

A straight line program of length n is a sequence of n assignment statements of the form

$$x_1 \leftarrow f_1(y_{11},\ldots,y_{12})$$
$$\vdots$$
$$x_n \leftarrow f_n(y_{n1},\ldots,y_{n2})$$

where the x_i and y_{ij} are (not necessarily distinct) variables and the f_i are (not necessarily distinct) k-any operations. Those variables which never appear on the left hand side of an assignment are called input variables.

Let us assume that each value for a variable occurring during the execution of the straight line program can be stored in one register and that given y_{i1} through y_{ik}, $f(y_{i1},\ldots,y_{ik})$ can be computed using a fixed number of registers. Holding k

fixed, we can ask how many auxiliary cells, i.e. cells other than those which contain the values of the input variables, are needed to execute a straight line program of length n.

We claim that O(n/logn) auxiliary cells are sufficient. This can be seen as follows:

Modify the program such that no variable appears twice on the left hand side of an assignment, hereby possibly increasing the number of distinct variables used. Construct a graph G as follows: the vertices of G are the distinct variables of the modified program. There is an edge $v_1 \rightarrow v_2$ iff in one line v_2 occurs on the left hand side and v_1 appears on the right hand side. G is directed, acyclic, has fan in k and has at most kn vertices and edges.

By Lemma 1 there exists a pebble strategy for G using at most O(n/logn) pebbles, which induces an evaluation strategy for the straight line program in an obvious way. (Unfortunately this strategy may be rather time consuming).

This result stands in perfect analogy to the fact, that each formula of length n can be evaluated with O(logn) auxiliary cells, because each k-ary tree with n leaves can be pebbled with O(logn) pebbles.

References:

1. Aho, A.V., J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

2. Cook, S.A., "An Observation of Time-Storage Trade-Off", ACM-STOC, 1973, 29-33.

3. Hartmanis, J. and R.E. Stearns, "On the Computational Complexity of Algorithms", Trans. AMS 117, 1965, 285-306.

4. Hennie, F.C., "On-line Turing Machine Computations", IEEE-EC 15, 1966, 35-44.

5. Hennie, F.C. and R.E. Stearns, "Two-tape Simulation of Multitape Turing Machines", JACM 13:4, 1966, 533-546.

6. Hopcroft, J.E. and J.D. Ullman, "Some Results on Tape-Bounded Turing Machines", JACM 16, 1967, 168-177.

7. Hopcroft, J.E. and J.D. Ullman, Formal Languages and their Relation to Automata, Addison-Wesley, 1969.

8. Paterson, M.S., "Tape Bounds for Time-Bounded Turing Machines", JCSS 6, 1972, 116-124.

9. Paterson, M.S., and L.G. Valiant, "Circuit Size is Nonlinear in Depth", Theory of Computation, Report 8, University of Warwick 1975.

10. Savitch, W.J., "Relationships between Nondeterministic and Deterministic Tape Complexities", JCSS 4:2, 1970, 117-192.

11. Stearns, R.E., J. Hartmanis, and P.M. Lewis, "Hierarchies of Memory Limited Computations", IEEE-SWAT, 1965, 191-202.