On Time Versus Space

JOHN HOPCROFT*

Cornell University, Ithaca, New York

WOLFGANG PAUL

Cornell University, Ithaca, New York

AND

LESLIE VALIANT

University of Leeds, 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 cannot be recognized in linear time by deterministic multitape Turing machines

KEY WORDS 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 multitape Turing machines. We establish for all functions $t(n)$ that the class of sets accepted in time $t(n)$ is contained with 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 cannot be recognized in linear time on a deterministic multitape 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 multitape 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) multitape Turing machines of time complexity $t(n)$. Let DSPACE($s(n)$) (NSPACE($s(n)$)) be the class of sets accepted by deterministic (nondeterministic) multitape Turing machines of space complexity $s(n)$.

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

Authors' present addresses: J Hopcroft, Department of Computer Science, Cornell University, Ithaca, NY 14853; W. Paul, Fakultat Mathematik, Universitat Bielefeld D-48, Bielefeld, Germany, L Valiant, University of Leeds, Leeds, Great Britain.

We present an algorithm for simulating a deterministic multitape Turing machine of time complexity t by a deterministic Turing machine of space complexity $t/\log t$. 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 $|w_1| = t^{2/3}$. 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	w_4^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 $(k+1)\, t^{1/3} \log t$ since approximately $\log t$ 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. 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 them 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 always 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 $O(n/\log n)$.

LEMMA 1. *For each k, $P_k(n)$ is $O(n/\log n)$.*

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 $O(n/\log n)$.

Let

$G = (V, E)$ be a graph in G_k with $R_k(n)$ edges which requires n pebbles,

$V_1 =$ the set of vertices of G to which a pebble can be moved using $n/2$ or fewer pebbles,

$V_2 = V - V_1$,

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

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

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

$A = E - (E_1 \cup E_2)$, 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 $n - 1$ 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 a subgraph with at least $(1/k)n/4$ fewer edges which requires at least $n/2$ pebbles. (To see this, note that the graph must have a vertex v of outdegree 0 which requires $3n/4$ pebbles. Vertex v must have an ancestor which requires at least $3n/4 - k$ pebbles. Thus we can delete v and the edges into v. The resulting graph still requires at least $3n/4 - k$ pebbles. Repeating the process $(1/k)n/4$ times, we obtain a subgraph with at least $(1/k)n/4$ fewer edges which requires at least $n/2$ pebbles.)

Thus in both cases $R_k(n) \geq 2R_k(n/2 - k) + (1/k)n/4$. Solving this recurrence gives $R_k(n) \geq cn \log n$ for some constant c This proof was inspired by [9]. □

Recently it has been shown by Paul, Tarjan, and Celoni (personal communication) that $P_2(n) \geq cn/\log n$ for some constant c, and hence Lemma 1 is optimal. This improves on an earlier bound of $c\sqrt{n}$ given by Cook [2].

LEMMA 2. $DTIME(t) \subseteq NSPACE(t/\log t)$.

PROOF. Let M be a $t \log t$ 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', 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 by 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', we can have M' simulate 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 is greater than or equal to 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 xth move$\}$. By (2.2) storing $c(\Delta)$ for one Δ takes space $O(t^{2/3})$. Thus if M' happens to guess a strategy which uses at most $O(t^{1/3}/\log t)$ pebbles at a time (by Lemma 1 such a strategy always exists), the simulation can indeed be carried out in space $O(t/\log t)$, 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 proved 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 $v(\Delta - 1) \rightarrow v(\Delta)$. 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 *first* move in a strategy which, starting from D, moves a pebble on the output vertex of G never using more than $O(t^{1/3}\log t)$ 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 $O(t^{1/3}\log t)$. Using techniques from [10], one can make it deterministic in space $O(t^{2/3}\log^2 t)$. 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 $DTIME(t) \subseteq DSPACE(t/\log t)$.*

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

COROLLARY 1. *For all t: $DTIME(t) \subseteq DSPACE(t/\log t)$.*

PROOF. Instead of precomputing t, successively try simulation of the proof of Theorem 1 for $t(n) = 1, 2, 3, \ldots$ until you 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 $t/(\log \delta)^{1/2}$ so that the number of blocks, and hence the size of the graph, is $(\log \delta)^{1/2}$. Then the total number of graphs is bounded by $\delta^{1/2}$, allowing us to cycle through all graphs contributing at most a multiplicative time factor of $\delta^{1/2}$. For a fixed graph we can construct a pebbling strategy in time δ. (There are at most $2^{(\log \delta)^{1/2}}$ configurations of pebbles, and hence nondeterministic space $(\log \delta)^{1/2}$ is sufficient. By Savitch's construction [10] deterministic space $\log \delta$ and hence deterministic time δ is sufficient.) Given a graph and a pebbling strategy, simulation requires time equal to the product of the number of pebble moves and $t/\log \delta$ or $(\delta)^{1/2}t/\log \delta$. Therefore the total time is bounded by $\delta^{1/2}[\delta + \delta^{1/2}t/\log \delta]$ or $\delta(n)t(n)$. Thus each k-tape machine of time complexity t can be simulated by a $(k+1)$-tape machine of time complexity less than $\delta(n)t(n)$ and tape complexity $O(t/\log\log \delta)$. Now an appeal to [5] yields the desired result.

It is an interesting open problem whether $NTIME(t) \subseteq NSPACE(t/\log t)$. 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

$$x_1 \leftarrow f_1(y_{11}, \ldots, y_{1k}),$$
$$\vdots$$
$$x_n \leftarrow f_n(y_{n1}, \ldots, y_{nk}),$$

where the x_i and y_{ij} are (not necessarily distinct) variables and the f_i are (not necessarily distinct) k-ary 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 $O(n/\log n)$ auxiliary cells are sufficient. This can be seen as follows: Modify the program so that no variable appears twice on the left-hand side of an assignment, thereby 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, is 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 $O(n/\log n)$ 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 $O(\log n)$ auxiliary cells because each k-ary tree with n leaves can be pebbled with $O(\log n)$ pebbles.

REFERENCES

(Note. References [1, 3, 4, 6, 8] are not cited in the text.)

1. Aho, A.V., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms* Addison-Wesley, Reading, Mass., 1974
2. Cook, S.A. An observation on time-storage trade-off. Proc. Fifth Annual ACM Symp. on Theory of Comput., 1973, pp. 29–33
3. Hartmanis, J., and Stearns, R.E. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. 117* (1965), 285–306
4. Hennie, F.C. On-line Turing machine computations. *IEEE Trans. Electronic Computers EC-15* (1966), 35–44
5. Hennie, F.C., and Stearns, R.E. Two-tape simulation of multitape Turing machines. *J. ACM 13*, 4 (Oct. 1966), 533–546
6. Hopcroft, J.E., and Ullman, J.D. Some results on tape-bounded Turing machines. *J. ACM 16*, 1 (Jan. 1969), 168–177
7. Hopcroft, J.E., and Ullman, J.D. *Formal Languages and Their Relation to Automata*. Addison-Wesley, Reading, Mass., 1969
8. Paterson, M.S. Tape bounds for time-bounded Turing machines. *J. Computer System Sci. 6* (1972), 116–124
9. Paterson, M.S., and Valiant, L.G. Circuit size is nonlinear in depth. Theory of Comput. Rep. 8, U. of Warwick, Coventry, England, 1975, to appear in *Theoretical Computer Sci.*
10. Savitch, W.J. Relationships between nondeterministic and deterministic tape complexities. *J. Computer System Sci. 4*, 2 (1970), 117–192
11. Stearns, R.E., Hartmanis, J., and Lewis, P.M. Hierarchies of memory limited computations. Proc. Sixth IEEE Symp. on Switching and Automata Theory, 1965, pp. 191–202.

RECEIVED NOVEMBER 1975, REVISED MAY 1976