Information Processing Letters 21 (1985) 87–91
North-Holland

16 August 1985

SQUARE TIME IS OPTIMAL FOR SIMULATION OF ONE PUSHDOWN STORE OR ONE QUEUE BY AN OBLIVIOUS ONE-HEAD TAPE UNIT

Paul M.B. VITÁNYI

Centre for Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

Communicated by K. Mehlhorn
Received 26 March 1984
Revised 16 October 1984

To simulate a pushdown store or queue on-line by an oblivious one-head tape unit takes at least square time. Since each multitape Turing machine can be trivially simulated by an oblivious one-head tape unit in square time, this result is optimal.

Keywords: Multitape Turing machines, pushdown stores, queues, time complexity, square lower bounds, on-line simulation by oblivious single-head tape units, Kolmogorov complexity

1980 Mathematics Subject Classification: 68C40, 68C25, 68C05, 94B60, 10-00

1982 CR Categories: F.1.1, F.1.3, F.2.3

1. Introduction

Each multitape Turing machine can be simulated by an *oblivious* one-head tape unit in square time. Such a simulation is essentially the same as the obvious one by *nonoblivious* one-head tape units in [3]. For simulation by an oblivious one-head tape unit we derive a lower bound which matches this upper bound on the simulation time. So, in this case the obvious simulation is also optimal. The best previous lower bound on this simulation time was $n^{1.618}$ in [8]. Recall that in an *oblivious* Turing machine the movement of the storage tape heads is independent of the input, and is a function of time alone (see, for instance, [9]). We also assume that the steps at which the input is polled are the same for all input streams. It is obvious that we can simulate one device like a queue or a pushdown store by an *oblivious* one-head tape unit in time T(n) iff we can simulate a multiqueue or a multipushdown store machine that way [9]. Thus, it suffices to show the lower bound on the time to simulate one queue or one pushdown store. The proof uses Kolmogorov Complexity as [8] and some references in that paper. We are only interested in the storage structure of the devices, and consider them as transducers connected with an input terminal and an output terminal. A machine A simulates a machine B on-line in time T(n) if, for $t_1, t_2, \ldots, t_n$, the sequence of times at which B reads from the input terminal or writes to the output terminal, there are corresponding times $t'_1, t'_2, \ldots, t'_n$ at which A reads or writes the same symbols and $t'_i \leqslant T(t_i)$ for all $1 \leqslant i \leqslant n$. We use 'simulation' in the strong sense of 'on-line simulation' throughout. A one-head tape unit is used as synonym for a Turing machine with one single-head storage tape.

Note: Mutually independent work in [6,5,10] has recently shown (the greatest common intersection of the three) that square time is optimal for the on-line simulation of 2 tapes by 1 tape, also if the simulating machine is allowed to be nonoblivious.

2. Kolmogorov Complexity

The ideas on descriptional complexity below were developed independently by Kolmogorov [4] and Chaitin [1]. We follow [7]. Consider the problem of describing a string x over 0's and 1's. Any computable function f from strings over 0's and 1's to such strings, together with a string y, such that $f(y) = x$, is such a description. A descriptional complexity K_f of x, relative to f and y, is defined by

$$K_f(x|y) = \min\{|d| \mid d \in \{0,1\}^* \& f(dy) = x\}.$$

For the *universal* computable partial function f_0 we have that, for all f with appropriate constant c_f, for all strings x, y, $K_{f_0}(x|y) \leqslant K_f(x|y) + c_f$. So, the canonical relative descriptional complexity K(x, y) can be set equal to $K_{f_0}(x|y)$. Define the *descriptional complexity* of x as $K(x) = K(x|\epsilon)$. (ϵ denotes the empty string.) Since there are 2^n binary strings of length n, but only $2^n - 1$ possible shorter descriptions d, it follows that $K(x) \geqslant |x|$ for some binary string x of each length. We call such strings *incompressible*. It also follows that $K(x|y) \geqslant |x|$ for some binary string x of each length. As an illustration, a string x = uvw can be specified by v, $|x|$, $|u|$ and the bits of uw. Thus,

$$K(x) \leqslant K(v) + O(\log|x|) + |uw|,$$

so that with $K(x) \geqslant |x|$ we obtain

$$K(v) \geqslant |v| - O(\log|x|).$$

3. The square lower bound

Without loss of generality we assume that the tape units below have semi-infinite tapes. That is, the squares of the tapes can be enumerated from left to right by the natural numbers. The 0th square is called the *start* square. Assume further, also without loss of generality, that the tape units write only 0's and 1's in the storage squares and relax the *real-time* requirement to *constant delay*. A computation is of *constant delay* if there is a fixed constant c such that there are at most c computation steps in between processing the n*th* and the (n + 1)st input symbol, for all n. Thus, constant delay with c = 1 is the same as real-time, and it is not difficult to see that each computation of constant delay can be sped up to a real-time computation by expanding the storage alphabet and the size of the finite control.

Theorem. *The fastest on-line simulation of a pushdown store, or a queue, by an oblivious one-head tape unit takes* $\Theta(n^2)$ *time.*

Proof. A queue is real-time simulatable by some pushdown stores [2]. By the arguments mentioned in Section 1, therefore, the only thing we have to prove is the square lower bound for the simulation of one queue. So, let us consider a queue Q which can *store* 0's and 1's read from the input terminal and *unstore* the currently stored 0's and 1's one bit at a time by writing them to the output terminal, in a first-in-first-out fashion. Let M be an *actual* oblivious one-head tape unit simulating the *virtual* queue Q in time T(n). Without loss of generality, M has a semi-infinite tape and writes only 0's and 1's on its storage tape. Intuitively, we have detached Q from its input terminal and its output terminal and have replaced it by M which is programmed to behave as if it were Q. Thus, we can distinguish between M as the embodiment of Q, containing a 0–1 string as polled through the input terminal insofar as the front bits have not yet been written to the output terminal, and the actual encoding of Q's contents on M's storage tape. An *adversary demon* supplies the sequence of input commands. The adversary demon chooses input commands by observing the current actual state of machine M. It first determines the *rightmost* intersquare boundary B on the storage tape of M, such that M polls less than n input commands out of the first 2n input commands while its storage head is left of B. If at least $\frac{1}{4}n$ input commands are polled while the storage head scans the cell immediately right of B, then $T(n) \in \Omega(n^2)$ (cf. Case 1 below). Therefore, at least $\frac{3}{4}n$ input commands are polled while the head is left of B, and at least n input commands with the head right of B. (Recall that the head movement is the same for all sequences of input commands since M is oblivious.) Divide the tape into three segments

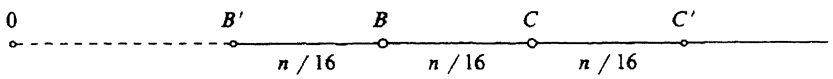

Fig. 1.

[0, B), [B, C), and [C, ∞), with the length of the middle segment [B, C) being $\frac{1}{16}$n squares (see Fig. 1).

We will argue that if $T(n) \notin \Omega(n^2)$, then we could describe an incompressible binary string in significantly less bits than its length, by reconstructing it from a description of M's behavior in simulating Q.

Case 1. Assume that at least $\frac{1}{4}$n input commands are polled with M's head on the middle segment [B, C). Consider a sufficiently long incompressible string $x \in \{0, 1\}^*$ of length 2n and let the first 2n input commands, supplied by the adversary demon, be 'store next unread bit of x'. Choose (p, q) as a pair of squares with p in [B', B) and q in [C, C'), where the lengths of both [B', B) and [C, C') are $\frac{1}{16}$n.

In the description of x below we give most of x in concatenated literal form in a suffix v and the concatenated remainder w of x in terms of M's operation. (Here x is a *shuffle* of v and w.)

- A description of this discussion in O(1) bits.
- A description of M in O(1) bits.
- The value of n in O(log n) bits.
- The location of the pair of squares (p, q) in O(log n) bits.
- The crossing sequence at that pair (p, q) of squares, as described below.
- The *final* contents of the tape segment [p, q], after the 2n input commands have been executed, thus storing all of x in the virtual queue Q (actually on M's storage tape).
- The concatenated literal remainder v of x in not more than $\frac{7}{4}$n bits.

For a pair (p, q) of squares on M's storage tape, as defined above, and the considered 2n-length sequence of input commands, the *crossing sequence* associated with that pair contains for each crossing the state of M, and whether we enter/exit [p, q] from/to left or right, in O(1) bits. Associated with each *entrance* of [p, q] we give the number of input commands polled up to the corresponding *exit* of [p, q]. Summed over all of the crossing sequence this does not take more than $O(\ell \log(n/\ell))$ bits, where ℓ is the length of the crossing sequence.

We can recover x from this description since it completely determines the unique instantaneous description of M in which feeding the successive bits of x through the input terminal drives M. Recovery of x goes as follows. The machine M is specified in the description. Beginning with M in the start state, enter bit for bit the literal representation v through the input terminal in M until the head of M enters [p, q]. Stop entering bits of v and continue by entering a 0–1 string of the proper length (the number of input commands prescribed in (p, q)'s crossing sequence) which leads from M's current instantaneous description to the correct exit of [p, q] (state of M and proper side). Starting from M's instantaneous description at that exit, continue inputting the remainder of the literal representation v, until [p, q] is entered again, and so on. Finally, after having input all-in-all 2n bits in M, which includes all of v, match the final contents of [p, q] with that from the description above. If there is a mismatch with the description above at any time during this process then backup and try other choices of proper length 0–1 sequences as input for the parts of the computation with M's storage head residing on [p, q]. There must be at least one 2n-length string satisfying the description since x does so by definition. Moreover, there is at most one 2n-length string satisfying the description. For, suppose there were two different 2n-length strings x and y which by the procedure above drive M into the same instantaneous description. (x and y can therefore only disagree on the bits received with M's storage head on [p, q].) Then, either we retrieve x having entered y or vice versa, by unstoring the entire present contents of queue Q as simulated by M: A contradiction. So the final i.d. of M resulting from the above exhaustive search, by inputting a string y which satisfies the above description all the way, must store x such that it can be retrieved by 2n 'unstore current front bit of simulated Q' com-

mands and $y = x$. Let the minimal length of any crossing sequence for a pair (p, q) be m(n). Then the description of x need take not more than

$$O(1) + O(\log n) + O(m(n) \log(n/m(n))) + \tfrac{3}{16}n + \tfrac{7}{4}n$$

bits. Since this amount must be at least $K(x) = 2n$, it follows that we have $m(n) \log(n/m(n)) \in \Omega(n)$ and, therefore, $m(n) \in \Omega(n)$. Summing the lengths of the crossing sequences of a set S of all pairs (p, q), with p in [B′, B) and q in [C, C′) and if $(p_1, q_1), (p_2, q_2) \in S$, then $p_1 \neq p_2$ and $q_1 \neq q_2$ must give a lower bound on the running time of M. Therefore, $T(2n) \geq (\frac{1}{16}n)m(n)$. Hence, $T(n) \in \Omega(n^2)$.

Case 2. Not more than $\frac{1}{4}n$ input commands are polled with M's head on [B, C). Therefore, out of the first 2n input commands, more than $\frac{3}{4}n$ input commands are polled with M's head on [0, B) and more than $\frac{3}{4}n$ input commands with M's head on [C, ∞). Let y be an incompressible binary string of length $\frac{3}{4}n$. The input is now supplied by the adversary demon as follows.

- The input commands polled with M's storage head left of B consist of storing the next unread bit of y on the simulated queue Q.
- The input commands polled with M's storage head right of C consist of unstoring the current front bit of the simulated queue Q. In case Q is empty, the input command polled is the 'skip' instruction which does not change anything to Q.
- The input commands polled with M's storage head on [B, C) are 'skip' instructions too.

This input strategy is followed by the adversary demon if at least one-half of the initial $\frac{3}{4}n$ store commands, polled with M's head on [0, B), in effect store bits of y on the simulated Q which are subsequently unstored by a corresponding one-half of the $\frac{3}{4}n$ unstore commands polled with M's head on [C, ∞). Else, the roles of [0, B) and [C, ∞) in the above input strategy of the demon and in the sequel of the argument are reversed. The argument then proceeds symmetrical. That these two possibilities are exhaustive is argumented below.

It is immediately clear that given y the demon must have used all y within 2n input commands.

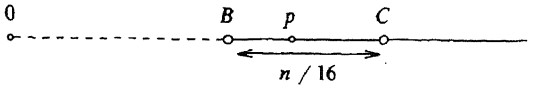

Fig. 2.

Also within 2n input commands, at least $\frac{3}{8}n$ bits of y polled with M's head on [0, B) have been retrieved by 'unstore front bit of Q' commands polled with M's head on [C, ∞), or vice versa under the alternative strategy of the demon. (Consider the subsums of the initial i elements of a 2m-length sequence consisting of m elements "+1" and m elements "−1". Either the number of i's, such that the sum from the 1st up to the *ith* element is greater than 0, is at least $\frac{1}{2}m$ or the number of i's, such that the sum from the 1st up to the *ith* element is smaller than 0, is at least $\frac{1}{2}m$. From this it easily follows that, in each particular sequence of m '*store-single-bit*'s and m '*unstore-single-bit*'s, either there must be at least $\frac{1}{2}m$ bits stored which are subsequently unstored or this holds for that sequence with the role of 'store' and 'unstore' everywhere interchanged.)

In the description of y below we give part of y literally in a suffix v and part w of y in terms of M's operation. Now v is a *concatenation* of v and w, that is $y = wv$.

- A description of this discussion in O(1) bits.
- A description of M in O(1) bits.
- The value of n in O(log n) bits.
- The location of B, C and a square p on [B, C) in O(log n) bits.
- The crossing sequence at p.
- The literal suffix v of y in not more than $\frac{3}{8}n$ bits.

The crossing sequence associated with p contains for each crossing the state of M and with each entrance of [0, p) the number of input commands for the simulated Q polled up to the corresponding exit of [0, p). Similar to Case 1, if $\ell(p)$ is the length of the crossing sequence at p, then the crossing sequence can be denoted in not more than $O(\ell(p) \log(n/\ell(p)))$ bits. Let the minimum length of such a crossing sequence in [B, C) be m(n). Then the description of y takes not more than

$$O(1) + O(\log n) + O(m(n) \log(n/m(n))) + \tfrac{3}{8}n$$

bits. To recover y from this description we proceed similarly to Case 1. We try out on M candidates for the 2n-length sequence of input commands for Q as described above. These sequences are determined by the demon's strategy and are appropriately interspersed with proper length sequences of input commands received while M's storage head resides on [0, p], containing commands which consecutively store the single bits of y on Q and possibly also 'skip' commands. Input commands polled with M's head on [C, ∞) constitute all of the unstore-front-bit (of Q) instructions and are supposed to yield the consecutive bits of y. To that purpose, attach a special register to M's finite control which remembers the last answer to such a query. This in case the answer to the query is output by M with its head *left* of p on [0, p) (or *right* of p on [p, ∞) under the alternative strategy of the demon). This slightly enlarges M's finite control, but leaves the preceding discussion intact. Thus, by extracting the consecutive bits from this register just before each query, while M's head is right of p, we must obtain at least the first $\frac{3}{8}n$ bits of y. The $\frac{3}{8}n$-length suffix v is given literally in the description. Clearly, y satisfies the description above. If there were different z and y satisfying the description, then either z would be retrieved by the commands polled with M's head on [C, ∞) while y was stored by the commands polled with M's head on [0, B), or y would be retrieved while z was stored: contradicting that M simulates Q. Since $K(y) = \frac{3}{4}n$ we have that $m(n)x \cdot \log(n/m(n)) \in \Omega(n)$. Therefore, $m(n) \in \Omega(n)$. Minorizing the running time T(2n) of M by summing the lengths of the crossing sequences over all squares of [B, C) to at least $m(n)(\frac{1}{16}n)$ we obtain $T(2n) \in \Omega(n^2)$. □

References

[1] G.J. Chaitin, Algorithmic information theory, IBM J. Res. Develop. 21 (1977) 350–359.

[2] P.C. Fischer, A.R. Meyer and A.L. Rosenberg, Real-time simulation of multihead tape units, J. Assoc. Comput. Mach. 19 (1972) 590–607.

[3] J.E. Hopcroft and J.D. Ullman, Formal Languages and Their Relations to Automata (Addison-Wesley, Reading, MA, 1969).

[4] A.N. Kolmogorov, Three approaches to the quantitative definition of information, Problems in Information Transmission 1 (1) (1965) 1–7.

[5] M(ing) Li, On one tape versus two stacks, Manuscript, Computer Science Dept., Cornell Univ., 1984.

[6] W. Maass, Quadratic lower bounds for deterministic and nondeterministic one-tape Turing machines, 16th ACM Symp. on Theory of Computing (1984) 401–408.

[7] W.J. Paul, J.I. Seiferas and J. Simon, An information-theoretic approach to time bounds for on-line computation, 12th ACM Symp. on Theory of Computing (1980) 357–367.

[8] P.M.B. Vitányi, An $n^{1.618}$ lower bound on the time to simulate one queue or two pushdown stores by one tape, Tech. Rept. IW 245, Centre for Mathematics and Computer Science, Amsterdam, 1983; Inform. Process. Lett. 21 (3) (1985) in press.

[9] P.M.B. Vitányi, An optimal simulation of counter machines, SIAM J. Comput. 14 (1985) in press.

[10] P.M.B. Vitányi, One queue or two pushdown stores take square time on a one-head tape unit, Tech. Rept. CS-R8406, Computer Science Dept., Centre for Mathematics and Computer Science (C.W.I.), Amsterdam, 1984.