RELATIVIZED OBLIVIOUSNESS*)

(Extended abstract)

Paul M.B. Vitányi

Mathematisch Centrum

Amsterdam, The Netherlands

ABSTRACT

Relativized obliviousness is introduced to capture the intuitive idea, that some problems allow fastest computations which are more oblivious than do other problems, without any of such computations being oblivious in the standard sense. It is shown that each increase in the obliviousness of an algorithm (in several different well-defined meanings), for the solution of some problems, may necessarily require an increase in computation time from $T(n)$ steps to $T(n) \log T(n)$ steps. There is, however, no problem for which a total oblivious algorithm requires more than order $T(n) \log T(n)$ steps, if the best algorithm for it runs in $T(n)$ steps. We use on-line Turing machines as model of computation.

1. INTRODUCTION

An oblivious on-line Turing machine is one whose head movements are fixed functions of time, independent of the actual inputs to the machine. In this paper we introduce the notion of relativized obliviousness, to capture the nature of algorithms (and problems) which seem partly oblivious and partly not. The results show that a small difference in obliviousness between algorithms used for the solution of a given problem may incur an increase in running time which is as great as the penalty for using a completely oblivious algorithm.

The concept of an oblivious algorithm is interesting for several reasons. Just as a machine model provides a certain formalization of the idea of an algorithm, so does the notion of an oblivious machine provide a certain formalization for the notion of an oblivious algorithm. Apparently, the concept was first introduced by PATERSON, M. FISCHER and MEYER [1974] to capture the notion of an algorithm being independent of the actual data. For instance, a table look-up by sequential search can be programmed obliviously (reading to the end-of-table after having found the looked-for item), while a binary search cannot be, since the number of items examined is small compared to the entire table and which items are examined depend on the item sought. Oblivious algorithms have been considered in a growing number of papers, since they allow us easier to derive lower bounds on time complexity of such computations, or time-space trade-offs, for concrete problems like sorting, searching, multiplication

---

*) This paper is registered at the Mathematical Centre as report IW 137/80

of binary numbers, matrix inversion and so on. (See the recent conference proceedings of e.g. FOCS and STOC meetings.) However, there are, for non-oblivious algorithms, very often but a few places in the computation where nonoblivious behaviour is required; but inbetween these places the computation proceeds obliviously. Hence the machine performing the computation (and the nature of the problem it solves) is oblivious to certain parts or aspects of the problem presented. In the sequel we select from the possiblities which suggest themselves, to make the idea of relativized (or partial) obliviousness concrete, the following: obliviousness relative to a subset of Σ^*, where Σ is the input alphabet; obliviousness relative to a subset of Σ (throughout the input-string, in a sense to be defined); the degree k of nonobliviousness, where k is the least number of disjoint subsets in which Σ can be partitioned so that the computation proceeds oblivious relative to each such subalphabet; and finally a finite bound on the total number of nonoblivious moves the machine can make during the processing of the input. We indicate how these different notions of relativized obliviousness and degrees of nonobliviousness are related and derive the following main results.

For each $k > 1$ there is language O_k which can be recognized in real-time by a k-nonoblivious on-line Turing machine, but for any $k' < k$ the fastest on-line k'-nonoblivious Turing machine recognizes O_k in time $\Theta(n \log n)$.

For each $k > 0$ there is a language N_k which can be recognized in real-time by an on-line Turing machine which makes at most k nonoblivious moves during the processing of an input, but for any $k' < k$ the fastest on-line Turing machine making at most k' nonoblivious moves during the processing of an input recognizing N uses time $\Theta(n \log n)$.

This paper is an extended abstract of a preliminary investigation; complete proofs, additional results as well as justification of the naturalness of the chosen concepts by illustrating them in relation to some storage-retrieval problems will be given in a final version to appear elsewhere.

2. RELATIVIZED OBLIVIOUSNESS

We assume the reader to be familiar with the concepts of k-tape on-line deterministic Turing machines, real-time computations on such machines etc., as used by e.g. P. FISHER, MEYER and ROSENBERG [1972]. Recall, that such machines have a separate one-way read-only input tape, and a one-way write-only output tape, apart from the k storage tapes. This is the model of computation we shall use throughout the paper, and is intended by the unqualified use of the term "Turing machine", although the definitions and results below hold also for more sophisticated models such as multi-head Turing machines with jumps. We say that a Turing machine is *oblivious* if the movement of head i at step t, $i = 1,2,\ldots,k$ when we talk about a k-tape machine, depends only on i and t, for each storage tape head i. Likewise, the movements of the input tape head and output tape head at step t depend on t only. One may think of the head movements as being controlled by a second autonomous machine which has

storage tapes but no input or output tapes. In the introduction we mentioned some grounds to refine the notion of nonobliviousness by identifying large oblivious parts of a computation which is not oblivious altogether. Below we define several concepts of relativized obliviousness, and of measures of degrees of nonobliviousness, all of which definitions hold for each model of computation for which obliviousness is defined.

Let M be a Turing machine with input alphabet Σ. By grouping together equal length input words, which cause M to execute identical sequences of head movements (taking into consideration the movements of the input tape head, the storage tape heads, and the output tape head), M induces an equivalence relation $\equiv_M$ on Σ^*.

DEFINITION 1.

(i) $\varepsilon \equiv_M \varepsilon$.

(ii) $xa \equiv_M yb$, $x,y \in \Sigma^*$ and $a,b \in \Sigma$, if $x \equiv_M y$ and M makes exactly the same sequence of head movements from shifting its input tape head to a till just before it shifts its input tape head to the right of a, on an input word starting with xa, as it does from shifting its input tape head to b till just before it shifts its input tape head to the right of b, on an input word starting with yb.

(iii) For no $x,y \in \Sigma^*$ it holds that $x \equiv_M y$ if not by (i)-(ii).

It is easy to see that $\equiv_M$ is an equivalence relation on Σ^*, and that it can only hold between equal length words. In this paper we consider *on-line* computations only. In defining a similar notion for *off-line* computations, or to capture some more aspects of relativized obliviousness of on-line computations, we may need to add the following requirement to (ii):

(*) $\quad x \equiv_M y$ iff for all $z \in \Sigma^*$ holds $xz \equiv_M yz$.

This has the effect of turning $\equiv_M$ into a right congruence relation, and means that if $x \equiv_M y$ then the future head movements of M do not depend on whether M first processes x or y. Our main results, however, do not depend on whether or not restriction (*) is included in (ii), since they deal with the notion introduced in definition 3 below, which essentially is concerned with infinite words, and therefore is invariant under this restriction.

DEFINITION 2. A Turing machine M with input alphabet Σ is *oblivious relative to* W, $W \subseteq \Sigma^*$, if for all words $x,y \in W$, $|x| = |y|$, holds $x \equiv_M y$. For short we call such an M: W-*oblivious*.

DEFINITION 3. A Turing machine M with input alphabet Σ is *oblivious relative to the alphabet* Δ, $\Delta \subseteq \Sigma$, if

(i) h is a homomorphism $h: \Sigma^* \to (\{\phi\} \cup (\Sigma-\Delta))^*$ defined by $h(a) = \phi$ for all $a \in \Delta$ and $h(a) = a$ for all $a \in \Sigma-\Delta$;

(ii) for all $w \in \Sigma^*$, M is $h^{-1}h(w)$-oblivious.

For short we call such an M: Δ-*alphabet-oblivious*.

Note that alphabet-obliviousness is a weaker notion than the corresponding monoid obliviousness. Thus, if M is Δ-alphabet-oblivious, then M is also Δ^*-oblivious. But M may very well be Δ^*-oblivious without being also Δ-alphabet-oblivious for $\Delta \subset \Sigma$. We now relate the above defined relativized obliviousness to the earlier concepts.

- M is oblivious iff M is Σ-alphabet-oblivious iff M is Σ^*-oblivious, for Σ the input alphabet of M.
- If $\{a\}$ is a singleton subset of the input alphabet of M, then M is both $\{a\}$-alphabet-oblivious and $\{a\}^*$-oblivious.
- The input monoid Σ^* can contain infinitely many distinct subsets W_i, $i \in \mathbb{N}$, such that a given machine is W_i-oblivious for each $i \in \mathbb{N}$, but not W-oblivious for any $W \subset \Sigma^*$ such that $W_i \subset W$ for some $i \in \mathbb{N}$.
- The input alphabet Σ can contain at most $\#\Sigma$ subalphabets Δ_i such that a given machine is Δ_i-oblivious for each i, $1 \leq i \leq \#\Sigma$. This fact will form the basis for measuring degree of nonobliviousness below.

DEFINITION 4. A Turing machine M with input alphabet Σ has *degree of nonobliviousness* k, or is k-*nonoblivious*, if

(i) Σ can be partitioned into k disjoint nonempty subsets $\Delta_1, \Delta_2, \ldots, \Delta_k$, such that M is Δ_i-alphabet-oblivious for each i, $1 \leq i \leq k$;

(ii) Σ cannot be partitioned into $k' < k$ disjoint nonempty subsets $\Delta'_1, \Delta'_2, \ldots, \Delta'_{k'}$, such that M is Δ'_i-alphabet-oblivious for all i, $1 \leq i \leq k'$.

Hence every Turing machine M with input alphabet Σ has a degree of nonobliviousness between 1 (M is oblivious) and $\#\Sigma$ (that is, M is totally nonoblivious). PIPPENGER and M. FISCHER [1979] showed that any multitape Turing machine can be simulated on-line by an oblivious 2-tape Turing machine in time $O(n \log n)$ for n steps. They showed that this result cannot be improved in general, since there is a language L which is recognized by a 1-tape real-time Turing machine M, and any oblivious Turing machine M' recognizing L must use at least order $n \log n$ steps. Below we refine this result by showing that it holds for arbitrary small differences in degree of nonobliviousness. (The time complexity expressed is the *worst-case complexity*.)

THEOREM 1. *For each* $k > 1$ *there is a language* O_k *which can be recognized in real-time by a Turing machine* M_k *which is* k-*nonoblivious; any* k'-*nonoblivious Turing machine recognizing* O_k *has to use at least order* $n \log n$ *steps to do so in case* $k' < k$. *Moreover, for each* $k' < k$ *there are* k'-*nonoblivious Turing machine which recognize* O_k *in time* $O(n \log n)$.

PROOF SKETCH. First we define O_k. O_k is over the alphabet $\Sigma_k = \bigcup_{i=1}^{k} \Delta_i$ where $\Delta_i = \{a_i, \bar{a}_i\}$ for all i, $1 \leq i \leq k$.

O_k is defined in terms of a k-nonoblivious machine M_k which recognizes it in real-

time using k stacks in which each cell may contain a 0 or a 1. Initialize all k stacks to empty and the finite control to the start state. Start reading, one symbol at a step, the input word $s_1 s_2 \cdots s_i \cdots s_n$. At each step M_k processes the read input symbol as follows: (at the ith step M_k reads s_i)

(i) Say that the input symbol s_i M_k reads at the i^{th} step is in Δ_j ($1 \le i \le n$, $1 \le j \le k$), then this symbol s_i is pushed on all stacks h, $1 \le h < j$ and $j < h \le k$, as a 0 or a 1 subject to the following interpretation. The first symbol s_1 of the current input word $s_1 s_2 \cdots s_n$ is in this computation henceforth interpreted as a 1, and its counterpart in the subalphabet is hails from, say Δ_ℓ, is henceforth interpreted as 0. The first symbol M_k meets, subsequent to processing s_1 in the process of recognizing $s_1 s_2 \cdots s_n$, which is unequal to s_1, say $s \in \Delta_{\ell'}$, is henceforth interpreted as a 0 while its counterpart in $\Delta_{\ell'}$ is interpreted as a 1. For the remaining symbols in $\Sigma_k - (\Delta_\ell \cup \Delta_{\ell'})$ the unbarred symbols are interpreted as a 1 and the barred symbols as a 0.

(ii) M_k pops stack j. If the popped symbol was a 0 then M_k outputs a 0; if the popped symbol was a 1 then M_k outputs a 1; if the stack is empty then M_k outputs a 0.

The language O_k consists of those words $w \in \Sigma_k^*$, for which M_k outputs a 1 when it processes the last symbol of w.

CLAIM 1. M_k is k-nonoblivious, i.e., by the partition of Σ_k into $\Delta_1, \Delta_2, \ldots, \Delta_k$.

CLAIM 2. O_k is not recognized by any k'-nonoblivious Turing machine with $k' < k$ in time less than order n log n.

PROOF SKETCH OF CLAIM 2. Assume that O_k is recognized by a k'-nonoblivious Turing machine M with $k' < k$. Then there is a partition of Σ_k into disjoint nonempty subsets $\Gamma_1, \Gamma_2, \ldots, \Gamma_{k'}$, such that M is Γ_i-alphabet-oblivious for $i = 1,2,\ldots,k'$. Since Σ_k contains 2k elements, there must be a subset, say Γ_j ($1 \le j \le k'$), which contains at least 3 distinct letters, say s_1, s_2 and s_3. Now change M into a machine M^* recognizing $O_k \cap \{s_1 s_2 s_3 s_3\}\{s_1, s_2, s_3\}^*$ by checking for inclusion in $S = \{s_1 s_2 s_3 s_3\}\{s_1, s_2, s_3\}^*$ with the finite control. Since $k \ge 2$, either two out of s_1, s_2, s_3 hail from the same subalphabet $\Delta \in \{\Delta_i \mid 1 \le i \le k\}$ while the third comes from $\Sigma - \Delta$, or all 3 of s_1, s_2, s_3 come from distinct subalphabets $\Delta, \Delta', \Delta'' \in \{\Delta_i \mid 1 \le i \le k\}$. Hence we can select two elements, say s_1, s_2, which represent a push 1 and push 0 respectively on some stack in M_k, while the remaining s_3 represents a pop from that stack. Since M is by assumption $\{s_1, s_2, s_3\}$-alphabet-oblivious, on the input ensemble $\{s_1, s_2, s_3\}^*$ its head movements are independent of the received input symbols, but according to the pushing and popping regime of s_1, s_2 and s_3 it receives, it must store and retrieve information in an arbitrary and continuous manner. Using an elegant counting argument introduced by COOK and AANDERAA [1969], called an *overlap* argument, applicable to computations where heavy use is made continuously of previously read-in information, we can

prove that M^*, and hence M, must spend at least order n log n steps on inputs of length n in S.
END of Proof sketch of Claim 2.

Since PIPPENGER and FISCHER [1979] showed that each on-line Turing machine can be simulated on-line by a 2-tape oblivious Turing machine in time $O(n \log n)$, their result proves the last sentence of Theorem 1; and Claims 1 and 2 prove the first sentence. □

The reader will notice that we actually showed that no k'-nonoblivious Turing machine can on-line simulate certain aspects of k pushdown stores in less than order n log n time for k' < k. The whole result is perhaps more elegantly worded in terms of transducers or abstract storage units instead of on-line language recognizers. It would then read something like:
" There is an abstract storage unit consisting of k pushdown stores with a restricted set of possible commands, viz., pop stack j and push all other stacks ($1 \leq j \leq k$), which is k-nonoblivious. Each k'-nonoblivious abstract storage unit (Turing machine-like) which simulates it on-line must use at least order n log n time to do so in case k' < k ".

COROLLARY 2. *For each* k > 1 *and each* i ($1 \leq i < k$) *there is a* k*-nonoblivious Turing machine such that any* (k-i)*-nonoblivious Turing machine simulating it on-line must use at least order* n log n *steps for* n *steps of the former.*

COROLLARY 3. *Let* T(n) *be any time bound* $n \leq T(n) = o(n \log n)$ *(*$f = o(g)$ *means* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$*). The class of languages recognized in* DTIME(T(n))*, by multitape on-line Turing machines, contains an infinite proper hierarchy of language families, according to increasing degree of nonobliviousness of the fastest Turing machines accepting them.*

Another measure of degree of nonobliviousness is formed by bounds on the number of nonoblivious moves a machine is allowed to make during a computation. We may think of a machine which keeps count of the number of nonoblivious moves it makes, and, when that count exceeds a certain threshold, becomes oblivious. This measure of degree of nonobliviousness, although totally different from the preceding one, yields analog results, as shown below. One might therefore conjecture that such results hold for each (or many) meaningful measures of degree of nonobliviousness.

THEOREM 4. *For each integer* $k \geq 1$ *there is a language* N_k *which can be recognized by a* k*-tape real-time Turing machine* N_k *which makes* k *or less nonoblivious moves during each computation; any Turing machine which expends at most* k-1 *nonoblivious moves during each computation and recognizes* N_k *has to use at least order* n log n *time. Moreover, there is an oblivious Turing machine recognizing* N_k *in time* $O(n \log n)$.

PROOF SKETCH. We first define N_k over the alphabet {0,1,2}.

N_k consists of all strings xay2za such that $a \in \{0,1\}$, $xy \in \{0,1,2\}^*$, $z \in \{0,1\}^*$ and the following 2 conditions hold:

(i) The letter 2 appears in xay2 at most k times.

(ii) The length of z is equal to the length of y minus the number of occurrences of the letter 2 in y.

N_k is defined as follows:

N_k records the incoming bit-stream on all of its k stacks until the first 2 arrives. Then on the first stack N_k starts to pop and compare the popped symbol against the incoming symbol. If they are equal N_k outputs 1 otherwise 0. If the stack is empty N_k outputs 0. Meanwhile, on all remaining stacks N_k continues to push the incoming bits. When the second 2 arrives N_k starts similarly popping the second stack and comparing the popped symbol against the incoming symbol; meanwhile ignoring stack 1 and continueing the head movement there, and pushing all incoming bits on stacks 3 to k. And so on, for the 3^{rd} to k^{th} arriving letter 2. Therefore, N_k need make at most k nonoblivious moves in its computation, since it always rejects when it has seen k+1 letters 2.

The fact that the recognition of N_k by a Turing machine spending at most k-1 nonoblivious moves during its computation takes at least n log n steps is proven by induction on k. For k = 1 the theorem can be proved by applying an overlap argument similar to the one hinted at in the proof sketch of the previous theorem. For $k = j > 1$ we can show that we can reduce the problem either to the truth of the theorem for k = 1 or the truth of the theorem for the case k = j-1, both of which are true by induction assumption. The last sentence of the theorem follows as before. □

Because of the above Theorem 4, Corollaries 2 and 3 also hold with the concept of "k-nonobliviousness" replaced by "number of nonoblivious steps k" for each k. By the nature of the concept of k-nonobliviousness, a language over a finite alphabet cannot be inherently ∞-nonoblivious. However, no such natural restriction holds for the measure of the number of nonoblivious steps in a computation.

THEOREM 5. *There is a language* N *which is recognizable by a real-time Turing machine but which, for each* $T(n) = o(n \log n)$, $n \leq T(n)$, *cannot be recognized by a* T(n)*-time bounded Turing machine with a finite bound on the number of nonoblivious steps it may make during a computation. However,* N *can be recognized by an oblivious Turing machine in time* $O(n \log n)$.

PROOF SKETCH. Define N as N_k without restriction (i), i.e., there is no restriction on the number of times 2 may appear in the xay-part of a word. It is easy to see that N can be recognized by a multihead real-time Turing machine with head-to-head jumps; SAVITCH and VITÁNYI [1977]. KOSARAJU [1979] has shown that these devices can be simulated on-line in real-time by multitape Turing machines. Hence N is recognizable by a real-time Turing machine. By PIPPENGER and FISCHER's [1979] result it is

recognizable by an $O(n \log n)$ time bounded oblivious Turing machine. Since $\bigcup_{k=1}^{\infty} N_k = N$, it follows from Theorem 4 that any Turing machine which is allowed but a finitely bounded number of nonoblivious steps, need use at least order n log n time to recognize N. □

Since N_k is 2-nonoblivious for each k, and also N is 2-nonoblivious, we have that already each class of languages recognized by 2-nonoblivious Turing machines in time $T(n) = o(n \log n)$, $T(n) \geq n$, contains a whole infinite hierarchy as discussed, with respect the number of allowed nonoblivious steps, of T(n)-time-bounded Turing machine accepted language classes.

Yet another measure of bounded nonobliviousness to bound the number of nonoblivious steps as a function f(n) of the input length n.

REFERENCES

[1] COOK, S.A. & S.O. AANDERAA [1969], *On the minimum computation time of functions,* Trans. AMS 142, 291-314.

[2] FISCHER, P.C., A.R. MEYER & A.L. ROSENBERG [1972], *Real-time simulation of multihead tape units,* JACM 14, 590-607.

[3] KOSARAJU, R. [1979], *Real-time simulation of concatenable double-ended queues by double-ended queues,* Proceedings 11th ACM-STOC, 346-351.

[4] PATERSON, M.S., M.J. FISCHER & A.R. MEYER [1974], *An improved overlap argument for on-line multiplication,* SIAM-AMS Proceedings, Vol. 7, (Complexity of Computation), 97-112.

[5] PIPPENGER, N. & M.J. FISCHER [1979], *Relations among complexity measures,* JACM 26, 361-381.

[6] SAVITCH, W.J. & P.M.B. VITÁNYI [1977], *Linear time simulation of multihead Turing machines with head-to-head jumps,* Springer Lecture Notes in Computer Science (ICALP 4) 52, 453-464.