CHARACTERIZING AND REPRODUCING PROGRAM REFERENCING DYNAMICS

by

C. R. Dutt and D. Ferrari

Memorandum No. UCB/ERL M82/20

5 April 1982

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Characterizing and Reproducing Program Referencing Dynamics°

C.R. Dutt°° and D. Ferrari

Computer Science Division, EECS Department
and the Electronics Research Laboratory
University of California, Berkeley

ABSTRACT

A characterization of program referencing dynamics based on the temporal behavior of memory demand (as represented by the working set size of a program for a given window size) is proposed. A deterministic generative model which produces a reference string having a given dynamic characterization is then presented, and its practical implementation is discussed. An experimental study of the accuracy and the viability of such a model is performed, together with a theoretical and empirical investigation of the feasibility of constructing a synthetic program which produces an approximation to that artificial string, thereby exhibiting the given dynamic behavior. The results for working-set-like environments are good, those for other types of memory policies reveal that improvements to the string generation algorithm or different characterizations are needed.

Keywords: Dynamic characterization, Generative model, Memory demand, Program behavior, Program dynamics, Reference string generation, Referencing dynamics, Sampled working set, String reduction, Synthetic program, Virtual memory, Working set.

° The work reported in this paper was supported in part by the Computer Systems Design Program of the National Science Foundation under Grant MCS-8012900.
°° Current address: Bell Laboratories, Holmdel, NJ 07733.

1. Introduction

The need for characterizing the referencing behavior of programs arose with the advent of virtual memory systems, since the importance of this behavior was immediately recognized. Various models have been proposed and actually exploited so far (see for example [12]), but they have failed, in some way or another, to capture the dynamics of a program's referencing behavior, that is the aspect of such behavior which really influences a virtual memory system's performance. Indeed, this performance is determined not so much by the mean memory demands or by the mean page fault rates of the programs in the workload, but rather by the laws of variation of these indices in time and by the time relationships existing among the programs during their executions. Most of the existing models can successfully reproduce the mean values of program performance indices, but they cannot in general reproduce, simply because of the criteria their design has been based on, the dynamic variations of these indices.

This paper proposes a characterization of program dynamics and presents a model which is based on it. The power and the limitations of this model are discussed, and its use in the generation of reference strings and of synthetic programs obeying a given dynamic characterization is explored both theoretically and experimentally. The model is based on the variations in time of the memory demand of a program as represented by the working set size curve, and is therefore oriented towards representing program behavior in working-set-like environments.

The ability to reproduce a certain dynamic behavior by generating an artificial reference string which has a given characterization is quite useful in most applications of trace-driven simulation to the study of memory policies, of the interactions between memory policies and scheduling disciplines, and of program behavior. An artificial string is generally less expensive to procure, more easily modifiable in a controlled way, and less space-consuming than a real address trace. The last characteristic comes from the fact that in most cases an artificial string does not have to be stored between consecutive instances of its use, but can be produced by the generating program and processed on the fly every time it is needed. Furthermore, an artificial string can be designed and implemented according to given specifications (e.g., a characterization of its dynamics), thereby allowing one to perform completely controlled simulation experiments, which cannot be performed with a real string because of its lack of flexibility. A synthetic program exhibiting a given dynamic characterization is also useful when studies of the types mentioned above are to be empirically performed.

Section 2 introduces the proposed characterization, discussing it in the context of a paged virtual memory; this will be the context of our discussion throughout the paper, but the characterization and the resulting model could be extended to a segmented environment. An algorithm which is capable of producing a reference string with a given dynamic characterization is illustrated in Section 3. Both the characterization and the algorithm were first presented in [9], and refined versions of both in [10]. Section 4 describes the results of various experiments performed with a program that implements the algorithm discussed in Section 3 to explore its practical viability. The problem of building a synthetic program which, during its execution, will reference its virtual address space as dictated by a given dynamic characterization is examined in Section 5, where the results of some experiments performed with such a program are also reported.

2. A Characterization of Program Dynamics

In spite of all the efforts made so far, a viable approach to the characterization of the dynamic behavior of a program has not been proposed yet. At one extreme, we have a complete characterization, namely, the reference string generated by the program when processing a given set of input data (in this paper, by "behavior of a program" we shall always refer to the behavior of a program-input pair). The reference string (i.e., the sequence of addresses issued by the CPU during the program's execution) contains complete referencing dynamics information but is excessively long for most purposes, hard to classify and compare to other strings, inflexible (i.e., hard to modify in a controlled manner), and input-data dependent in unknown ways. At the opposite extreme, we find program behavior models whose dynamics either is totally unrelated to that of the original string (e.g., the Independent Reference Model [5]) or cannot be easily related to it (e.g., the two-level Markov-LRU Stack Model described in [6]).

The approach to be proposed here is based on the working set model. The appeal of this characterization stems from the attractiveness of that model, from the increasing popularity of working-set-like policies, and from the natural and relatively compact way in which the characterization can represent the dynamics of a program's memory demand. It should be immediately stressed that the characterization to be presented below, though based on the working set model and hence somehow biased towards variable-size memory policies of the working set type, is not intended only to describe the dynamic behavior of a program in a working set environment, but to model those aspects of such behavior which are the most relevant ones in any virtual memory environment. Whether or not this is indeed the case will be the subject of a discussion in Section 4.

Let virtual time be measured in memory references rather than in microseconds, and assume for simplicity that references are equally spaced on the virtual time axis. In this discrete-time context, the working set $W(t,T)$ at time t with window-size T can be defined as the set of the pages referenced in the closed interval $(t-T+1, t)$. If the reference string generated by the program is the finite sequence $r = \{r_t\}$ $(t = 1,2,...n)$, where r_t is the name of the page referenced at time t, the above definition of $W(t,T)$ can be extended to the interval $1 \le t \le T$ assuming that, for $2 - T \le t \le 0$, r_t is defined and is the name of a non-existing page whose size is 0. By $w(t,T)$ we shall denote the size of set W at time t with window size T.

A natural way to represent the dynamic behavior of a program is by using the curve $w(t,T)$ (see for instance [1] [3]), which, for a given window size T, is a function of virtual time t. Under the assumptions made in the previous paragraph, this characterization is totally equivalent to the one based on the string of integers $w = \{w_t\}$, $(t = 1,2,...n)$, which consists of the consecutive values of working set size. The objection that the amount of information in string w is as large as that found in the program's reference string can be answered by observing that the important aspects of a dynamic phenomenon can often be captured by assigning the coordinates of a relatively small number of points of the curves which describe it; also, string w may be modeled stochastically according to working set oriented criteria [8]; this question will be discussed further in Section 4.

However, the knowledge of $w(t,T)$ may not be sufficient to characterize a program's behavior. Not all of the arrivals and departures of pages cause working set size variations. While an increment in the working set size is always the result of the arrival of a new page (when running the program with those input data and that window size in a pure working set environment, this arrival would produce a page fault), there are arrivals which do not change the working set size. This is the case when the arrival of a new page is accompanied by the simultaneous departure of an old page from the working set. The above statement can be repeated for decrements in $w(t,T)$ and departures. Simultaneous arrivals and departures can be specified in several alternative ways; to simplify our discussion, we shall assume that the occurrence times of these events are given. All the other characterizations will have to lead to the string of the occurrence times in order for artificial reference strings with the assigned properties to be generated. Thus, the characterization we are proposing consists of the two finite sequences of integers

$$w = \{w_t\} \qquad (t = 1,2,...n),$$

$$f = \{f_j\} \qquad (j = 1,2,...k),$$

where f_j is the time index at which the j-th arrival-departure event occurs. For the sake of brevity, an arrival-departure event will be called a *flat fault* in the sequel, and the above will be called a *(w,f) characterization*. A reference string and its (w,f) characterization are presented in Fig. 1.

What conditions are to be satisfied by a (w,f) characterization in order for it to represent a reference string? Can w and f be assigned arbitrarily? Before answering these questions, we must introduce the departure set $D(t,T)$. First, we observe that a departure occurs at time t if either

$$w_t = w_{t-1} - 1$$

or

$$f_j = t$$

for some j. Second, we notice that in both cases the departing page is the one which has been referenced at time $t-T$. The *departure set* $D(t,T)$ is the set of pages which drop out of $W(t,T)$ during the closed interval $(t+1, t+T-1)$; in other words, it is the set of pages referenced between $t-T+1$ and t which are not referenced between t and $t+T-1$; its cardinality d_t at time t is equal to the sum of the number of working set size decrements and of the number of flat faults occurring between $t+1$ and $t+T-1$ (the page referenced at time t cannot drop out of W before $t+T$). Note that a *decrement* is said to take place at time $t+1$ if $w_{t+1}=w_t-1$. Figure 1 includes string $d=\{d_t\}$ $(t=1,2,...n)$ for $T=6$. Note also that, like W, set D can be defined for $t<T$ assuming that all references for $t\leq 0$ are made to a non-existing page whose size is 0, and which does neither arrive nor depart.

An answer to the previous questions can now be provided.

Theorem 1.

Given a reference string $r=\{r_t\}$ $(t=1,2,...n)$, and assuming that the working set of the program which has produced r is initially empty, its (w,f) characterization has the following properties:

(a) $w_1 = 1$;

(b) $0 < w_t \leq min(p,T)$ $(t=1,2,...n)$, where p is the total number of pages in the program;

(c) $f_1 > 0$, $f_j > f_{j-1}$ $(j=2,3,...k)$;

(d) $|w_t - w_{t-1}| \leq 1$ $(t=2,3,...n)$;

(e) $w_{f_j} = w_{f_j-1} < p$ $(j=1,2,...k)$;

(f) $d_t < w_t$ $(t=1,2,...n)$.

Proof.

(a) Since $W(t,T) = \emptyset$ for $t \leq 0$, after the first reference we have $w_1 = 1$.

(b) The working set, by definition, can never contain more than p pages or T pages, whichever is smaller. Also, its size must be positive at all times.

(c) The f_j's are positive time indices arranged in chronological order and corresponding to the occurrences of flat faults.

(d) By definition, the working set size after each reference can either change by 1 or remain constant. All other changes are impossible.

(e) A flat fault cannot occur when the working set size is either increasing or decreasing, or when it contains all the program's pages.

(f) Since no page may drop out of the working set more than once during an interval of T references, and the page referenced at time t is guaranteed to be in $W(t+T-1,T)$, the maximum cardinality of $D(t,T)$ equals $w_t - 1$ (see for example string d in Fig. 1). Q.E.D.

The inverse question now arises: given a (w,f) characterization which satisfies the conditions of Theorem 1, can one construct a reference string which is described by that

characterization? We shall see in the next section that this question has an affirmative answer.

3. The Generation of a Reference String Having a Given Dynamic Behavior

An algorithm will now be presented for obtaining a reference string with a given (w,f) characterization. It will then be shown that, if the characterization has the properties in Theorem 1, the algorithm completes its task successfully, and that, if it does not, the algorithm cannot generate the entire string.

The assigned dynamic behavior can be reproduced by referencing one of the pages not belonging to the working set whenever w or f indicate that an arrival is to take place, and one of the pages already in the working set otherwise. In addition, T references before the time a departure is expected to occur, the page which has just been referenced is to be marked to prevent further references to it until its departure time. If no departure is to take place T references from the current time t, the page just referenced will have to be referenced again on or before time $t+T$ (if it is referenced at $t+T$ for the first time after t, we shall consider it as a re-referenced page rather than viewing this as the departure and the simultaneous arrival of the same page).

In order to apply the algorithm just described, we need to know at any time t the current contents of the working set, whether an arrival must take place at t, and whether a departure must take place at $t+T$. Also, the pages in the working set are to be divided into two categories: the marked pages, which cannot be referenced before they drop out, and the unmarked ones, which must be referenced before they drop out. It is therefore convenient to construct and maintain three sets, the set of unmarked pages, to be called the *candidate set* C, the set of marked pages or *forbidden set* F, and the *external set* E, which contains all the pages not in the working set. These three sets are disjoint, and their union coincides with the set P of all pages. Thus,

$$C(t,T) \cup F(t,T) = W(t,T) = P - E(t,T).$$

To simplify our symbology, we shall leave the dependence of these sets on t and T implicit and write C,F, and E whenever no ambiguity may arise.

Page r_t, referenced at time t, will either join F or C, depending on whether or not its departure is expected to take place at $t+T$. In any case, the page will be assigned $t+T$ as its *time index*, and, when required, we shall write $r_t[t+T]$: if the page is added to F, the time index is the time at which it will leave F and join E, thereby departing from the working set; if it is added to C, the time index represents the latest possible time at which the page is to be referenced again in order for the given dynamics to be accurately reproduced. While the page to be referenced at time t can be chosen arbitrarily from among the members of C (if no arrival is to occur) or of E (if an arrival must take place), it may be convenient to keep the members of C ordered according to their time indices, i.e., in FIFO order, so as to minimize the probability of unwanted departures which will require repeating the string generation procedure for the last window with different page selections. On the other hand, the choice of a page from E does not have to satisfy any such condition and can be made according to several oriteria; for example, trying to reproduce given proportions of sequential and random referencing behaviors (one approach would be to assign the probability that the next external reference is to the next page, if indeed this page is not in the working set at that time). Set F does not have to be ordered, but the amount of searching to be performed at each reference to see whether the current time coincides with the time index of any member of F is minimized if F is treated as a FIFO queue.

The algorithm discussed informally so far in this section is more formally described in Fig. 2, where we have assumed that both C and F are FIFO queues. Note that in the figure *first* (Q) represents the oldest member of FIFO queue Q, and the union operator $\cup$ in $Q \cup \{x\}$ appends page x to FIFO queue Q. Note also that, if F is empty when Step 6 is executed, the test of the time index of its first element will have a negative result as if such element existed and its time index were different from t. Finally, whenever the procedure "error" is invoked, the algorithm halts, since it can no longer accurately reproduce the given dynamics.

A sample application of the algorithm in Fig. 2 is presented in Fig. 3. The given (w,f) characterization is the one of the reference string in Fig. 1. Step 5 of the algorithm clearly shows that, at all time instants, r_t is appended either to C or to F with time index $t+T$. This suggests that we can keep track of the contents of both C and F by writing the name of each referenced page on a single line (the C/F line in Fig. 3) T time instants ahead, i.e., at the time corresponding to its time index. When the first element of C is used as the next reference, its symbol on the C/F line is crossed out and rewritten T instants ahead. The elements of F, whose symbols appear underlined on line C/F in Fig. 3, are crossed out and join set E when the current time goes beyond their position (see Step 6). The generated string r^* does not, of course, coincide with the original string r but has the same dynamic properties, as characterized by the (w,f) pair. Note that the contents of set E are not shown in the figure.

We can now prove that the conditions of Theorem 1 are necessary and sufficient for the algorithm in Fig. 2 to generate a reference string with an assigned dynamic behavior.

Theorem 2.

The string generation algorithm described in Figure 2 produces a reference string with a given (w,f) characterization if and only if this characterization exhibits the properties (a) through (f) listed in Theorem 1.

Proof.

(A) The condition is necessary. Since by Theorem 1 all reference strings have properties (a) through (f), the given (w,f) characterization must have these properties in order for the generated string to be faithfully represented by it.

(B) The condition is sufficient. Let the given (w,f) characterization exhibit properties (a) through (f). The discussion of the algorithm in the first part of this section shows that, by construction, the algorithm generates a string r whose behavior is characterized by the given (w,f) description. This conclusion can also be reached by formally analyzing the reference selection phase (Step 3) and the set updating phase (Steps 5 and 6) in Figure 2: page arrivals are handled by referencing pages outside the working set (Step 3), page departures by preventing the referencing of the pages expected to leave (Steps 3 and 5) and by transferring them from F to E (i.e., out of the working set) at the proper times (Step 6). Thus, the only cases in which we cannot obtain the desired result arise when the algorithm is not allowed to terminate. There are three error exits in the description of Fig. 2, which will now be discussed separately.

(i) Let $w_t > w_{t-1}$ at some time t. If E were empty after the generation of r_{t-1}, we would have $w_{t-1} = p$, where p is the number of pages in the program. This would require $w_t > p$, which is impossible if condition (b) is satisfied. If on the other hand we have $f_j = t$ for some j, E cannot be empty, since we would have $w_{t-1} = p$, and condition (e) would not hold. Thus, the first error exit in Step 3 can never be taken if the given characterization has properties (b) and (e).

(ii) Let $w_t \leq w_{t-1}$ and $t \neq f_j$ for all j (i.e., no page arrival at time t). By definition, $D(t,T)$ is the set of pages leaving the working set between $t+1$ and $t+T-1$, and $F(t-1,T)$ is the set of departures expected to take place between t and $t+T-1$. Thus, if there is a departure at time t (i.e., if $w_t < w_{t-1}$), we have $|F(t-1,T)| = d_t + 1$; if, on the other hand, there is no departure (i.e., $w_t = w_{t-1}$), then $|F(t-1,T)| = d_t$. Since $W = C \cup F$, and C and F are disjoint, we have

$$w_{t-1} = |C(t-1,T)| + |F(t-1,T)|,$$

and, if $w_t < w_{t-1}$,

$$w_t < |C(t-1,T)| + d_t + 1.$$

If $C(t-1,T)$ were empty, condition (f) would not be satisfied, since we would have $w_t \leq d_t$.

Let us now assume $w_t = w_{t-1}$. Then

$$w_t = |C(t-1,T)| + d_t.$$

The sampling methodology adopted for obtaining relevant points of the $w(t)$ curve is based on two parameters δ and μ.

First, the total number of points n in the original string w is reduced by a factor of δ by sampling w at intervals of δ time units; the reduced string $w_\delta, w_{2\delta}, \cdots w_n$ will be denoted by w_τ. In our experiments, we set δ=50. Since the original string consisted of 500,000 references, the number of points was first reduced to 10,000. The importance of guaranteeing an interval of at least δ time units between two consecutive sample points will become clear in Section 5.

String w_τ is then sampled in the following manner:' two consecutive sample points $w_{i-\delta}$ and w_i, with $w_{i-\delta} \neq w_i$, are selected, provided that the time that elapsed since the selection of the last point exceeds some minimum value μ; otherwise, only w_i is selected. The motivation behind the introduction of parameter μ is primarily to test the accuracy of our interpolation technique in approximating the real $w(t,T)$ curve: we do not want to specify too many points that are very close to each other, but, rather let the interpolation fill in the curve. When, for example, many slope changes occur over a relatively short period of time, we would end up with a large number of points; by reducing this number, we not only save more space, but we also speed up the string-generation algorithm, which will have to deal with fewer input points. Of course, μ must not be too large, since fluctuations in the working set size over a large time interval would cause an intolerable amount of deviation between the interpolation approximation and the real $w(t)$ curve. For our experiments, we selected μ = 2000, and we obtained the impressive further reduction to 524 points.

We shall now try to determine what properties of a modeled program's referencing behavior are preserved in the behavior of the reference string artificially generated by the algorithm described in Section 3. Our approach will be the one schematically represented in Fig. 5.

The program to be modeled produces (with a given set of input data) a reference string r, which, processed by a pure working set policy with a given T, yields a (w,f) characterization of the program. This characterization is reduced as described above, and then input to GENWS for the generation of the artificial reference string r'. String r' is processed under a given memory policy, and a set of performance indices I' are obtained. String r is also processed under the same policy, and produces values of the same performance indices to be collectively denoted by I.

We want to investigate the following issues:

(1) Which of the performance indices I are reproduced (within a tolerable level of error) by the generated r'?

(2) How accurate is our interpolation technique in approximating the real $w(t,T)$ curve, and how sensitive are the performance indices to such an approximation?

(3) By how much can we reduce the number of points required for a complete characterization, and still obtain values for set I' reasonably close to the corresponding values for set I?

(4) Is the accuracy of the model sensitive to T?

One important observation should be made here: given a complete characterization of $w(t,T)$ by specifying the coordinates of all the points where a change of slope occurs, the model will reproduce the program's $w(t,T)$ curve exactly; thus, there is no reason to validate results for complete (w,f) characterizations, since, in that case, GENWS guarantees faithful reproduction of the working set size at all points, and we shall only consider incomplete characterizations.

The trace used in our experiments consisted of the first 500,000 references generated by a program written in APL during its execution. This trace is described also in [11]. Page addresses were obtained from the original virtual addresses in the trace by assuming a page size of 1024 bytes.

Figure 6 shows a plot of the $w(t,T)$ curve for our trace with T=10,000. This value of T allows us to distinguish various phases during the execution of the program. For

example, the steep slopes are characteristically (though by no means exclusively) produced by phases of sequential behavior, where the program references a large number of pages over a relatively short period of time; the long flat section (interrupted only by a "rectangular pulse") is characteristic of a local phase (typically produced by a loop) during which a relatively small number of pages are repeatedly referenced. Since this program contains aspects of both extremes of paging dynamics, we felt we could test the model over a wide range of referencing behaviors simply by using this single trace. Various published $w(t,T)$ curves with realistic values of T (see, for example, [1], [4], [11]) show patterns of behavior similar to those that appear in Fig. 6.

For T=10,000, no flat faults were found to occur in the string. This is hardly surprising, since, for T=10,000, a flat fault can occur at time t only if a page fault occurs at time t and if the page referenced at time $(t-10{,}000)$ is never referenced again before time t, a highly unlikely event. At T=5000, no flat faults were detected either. Only for much smaller, unrealistic window sizes were some flat faults found (for instance, 6 for T=1000). Thus, flat faults were ignored in all our experiments.

a. Experiments under WS and SWS with T=10,000 references

Having used T=10,000 references in the generation of artificial string r', in the first experiment following the general scheme of Fig. 5 the memory policy was the pure working set policy with T=10,000. The results are presented in Table I.

Note that $\bar{\delta}$ is defined as

$$\bar{\delta} = \frac{1}{n}\sum_{i=1}^{n} |w(i) - w'(i)| ,$$

where $w'(i)$ is the working set size of r' at time i. All results, including $\bar{\delta}$, which is a measure of dynamic fit between the two curves w and w', indicate that the approximation is an excellent one, in spite of the reduction by three orders of magnitude of the size of the input to GENWS. Since there are 850 points at which the slope of the original $w(t,T)$ curve changes, our input to GENWS, consisting of 524 points, represents a 38% reduction of the minimum quantity of information needed for a complete characterization of the curve. When $w'(t)$ is plotted on the same diagram as $w(t)$ (see Fig. 6), the two curves are practically indistinguishable, as are the two page fault plots shown in Fig. 7.

The results obtained under the sampled WS policy (SWS) with a sampling interval equal to the window size were very similar to those just presented. Note that the $w(t)$ curve sampled with our methodology was that generated by r in a SWS environment with T=10,000, and that only 305 points of that curve were sufficient to obtain a very accurate reproduction of the dynamic characterization of r.

b. Experiments under WS and SWS with T=5000 and T=20,000

When the string r' generated in the previous experiments was run in a simulated working set environment with T=20,000, the results were quite close to those obtained by running r in the same environment (the mean working set sizes and the mean page fault rates were practically the same, and the relative error in the space-time product was -8%), the only exception being the maximum working set size (56 pages for r', 78 pages for r). A totally different set of results was obtained, as expected, with T=5,000. The relative errors in the mean working set size, in the mean page fault rate, and in the mean space-time product were +5%, -40%, and -30%, respectively. No error was found, however, in the maximum working set size of r'. Similar results were obtained by running r and r' under the SWS policy with T=20,000 and T=5,000.

c. Experiments under local LRU with m=20 page frames

The size m of the fixed partition for our LRU experiment was chosen on the basis of Denning and Schwartz's observation [7] that, if T_m is the value of T at which $\bar{w}(T)=m$, then $\bar{f}_{LRU}(m)\approx\bar{f}_{WS}(T_m)$. That is, when the size of the partition equals the mean working set size, the page fault rates of LRU and WS are about the same (in reality, $\bar{f}_{LRU}(m)$ tends to be slightly higher than $\bar{f}_{WS}(T_m)$). Since, for T=10,000 references, $\bar{w}\approx$20 pages, we chose m=20 page frames. The mean fault rate produced by r under LRU was somewhat

higher than that produced by the same string under WS, as expected. However, the mean fault rate produced by r' under LRU turned out to be orders of magnitude greater than that produced by r under the same policy, in spite of the maximum-locality approach implemented by the algorithm in the version presented in Section 3. This can be explained by observing that, to keep $w(t)$ constant at some value w over an interval of time, GENWS will reference w pages cyclically, so a typical substring of r' might look like

$$\ldots 1,2,3\ldots w-1,w,1,2,3,\ldots w-1,w,1,2,3\ldots$$

Note that any two adjacent elements in such a substring are distinct, and that the interreference interval for any page (i.e., the interval of time between successive references to that page) is at least w. What happens is that, over an interval of time during which $w>m$, every consecutive reference in r' (after the initial m distinct pages have been loaded) will produce a page fault!

This problem could be solved or alleviated by allowing the user to specify, for example, the mean interreference times between pages for all pages in the program's address space. Thus, the algorithm, instead of generating pages in cyclic order, could attempt to preserve the values of these times by an appropriate selection of pages from the candidate set. (The only reason for treating the candidate set as a FIFO queue is to make the probability of unwanted departures equal to zero, thereby avoiding the need to check that these events did not occur.)

d. Experiments under PFF with τ=1543 references

The value of the PFF policy parameter τ [4] was obtained by setting it equal to the mean lifetime the program exhibits under the WS policy with T=10,000 references. As can be derived from Table I, the mean lifetime is 1543 references. With this value of τ, the program's mean memory occupancy is about equal to $\overline{w}$. The results of our experiment, performed according to the general scheme in Fig. 5, showed large differences between the values of the mean memory occupancies, space-time products, and page fault rates produced by r and r'. To explain the reasons for these results, we note that if the largest working set size that appears in the (w,f) characterization for r is w_{max}, then GENWS will produce r' using a set of only w_{max} distinct pages (which are identified by the integers $1,2,\ldots w_{max}$). In r', as soon as m reaches w_{max}, no page faults can occur again under PFF, simply because no new page numbers outside this set will be generated. (For T=10,000, for example, r' under PFF produces a constant $m=w_{max}$=56, and no page faults at all soon after time t=10,000.) In reality, a program's execution usually involves many more pages that the maximum working set size observed under a working set policy with a fixed T. Thus, the set of pages that GENWS uses should have the same size as that of referenced pages; the user could, for example, be allowed to specify the page reference probabilities.

Of course, the specification of interreference times and of probabilities for page references has the disadvantage that the model increases in complexity and becomes cumbersome to use.

An open question, to be investigated in a future paper, is whether the string-generation algorithm described in Section 3 can be extended so that it can generate a reference string r' that satisfies two (or more) $w(t,T)$ curves for two (or more) different values of T. In this case, would the resulting reference string be a significantly more accurate model of r (for memory policies different from WS and SWS) than an artificial string satisfying just one curve?

e. Experiments under WS with different nominal window sizes

To test the sensitivity of the model's accuracy to changes in the window size (denoted by T in Fig. 5), the experiment described in a above under the pure WS policy was repeated for T=5,000, 15,000, and 20,000 references. Some of the results are plotted in Fig. 8, which shows the space-time products of r (solid line) and r' (broken line) vs. T. As may be seen in Fig. 8, the errors in the space-time product for window sizes different from T=10,000 were larger than those for T=10,000 but always very small. The lower values of the space-time product for r' at all window sizes were due to the slightly fewer page faults produced by GENWS because of the bias of its interpolation method towards local behavior. The values of the dynamic error $\overline{\delta}$ ranged from 2.27% (of $\overline{w}$) for T=5,000 to

0.58% for T = 20,000. Thus, the dynamic accuracy of the reproduction improved as the variability of $w(t)$ decreased.

The reduction factors characterizing the incomplete specification of the w strings which were input to GENWS ranged between 41% for T=5,000 and 36% for T=20,000 with respect to the minimum-size complete specification. With respect to the amount of information needed to specify the whole w string, the incomplete specifications represented reductions by factors from 542 for T=5,000 to 1129 for T=20,000.

5. Reproducing Dynamics by an Artificial Program

We now turn our attention to the problem of constructing an artificial program which, given a reference string (for example, one generated by GENWS), reproduces during its execution the given string. This program, if it can be implemented, may be used to emulate the memory consumption pattern of an actual program in a computer system's workload.

The problem of constructing synthetic programs that reproduce characteristics of program referencing behavior for virtual memory environments has recently received some attention; in particular, Babaoglu's design [2] of a program that is able to reproduce a given lifetime curve has a number of aspects in common with the problem being discussed here.

In this section, we show how, with the aid of GENWS, a program may be designed which reproduces a given $w(t,T)$ curve while executing under a pure working set policy (with window size T), and that the program works well even in the more realistic environment of a sampled working set policy.

A possible approach to this problem, the one we shall actually take, is to use GENWS to generate a reference string r' having the given (w,f) characterization, and then to construct a synthetic program which, when running, issues a string of references ideally coinciding with r'. The program may be organized as shown in Fig. 9. The program reads a page address r_1 and then immediately references page r_1 itself, reads the next page address r_2 and immediately accesses page r_2, and so on. This typical "read-next-page-and-access" pattern, where the program "flies off" to access the desired page, and then "rebounds" back to the page containing the code of the artificial program to execute the next instruction, explains why the method has been called the *boomerang approach*, and the program it produces the "BOOMERANG" program. Unfortunately, with this procedure, although the desired pages are referenced during the execution of the program, there also are a number of undesired references occurring in between the desired ones, caused by the fetching and execution of BOOMERANG's own instructions, and by the accesses to the page that contains the segment of r' being currently referenced. The reference string r'' produced by BOOMERANG, then, is of the form $q_1q_2\cdots q_Qr_1q_1q_2\cdots q_Qr_2q_1q_2\cdots q_Qr_3\cdots$, where the r_i are the desired page references from r' and the q_i are extraneous references. Note that the r_i are assumed to be available at the outset and not to be computed on the fly because of the excessive number of extraneous references that would result from the on-the-fly computation approach.

All sequences of extraneous references $q_1q_2\cdots q_Q$ have the same length Q and reference the same number V of distinct pages. Thus, the extraneous references produce the following two effects on the (w,f) characterization of r'': (a) the time scale is expanded by a factor of $(Q+1)$ with respect to that of the characterization of r' (r'' is $(Q+1)$ times as long as r'), and (b) the working set sizes are all increased by V or slightly more than V, depending on the number of pages containing segments of r' accessed during one window.

We can reverse these effects simply by transforming the desired (w,f) characterization into a new characterization (w^*,f^*) to be used to generate a reference string r^* which, when input to BOOMERANG, causes the resulting r'' to exhibit the original (w,f) characterization (see Fig. 10). Such a transformation involves shrinking the time scale and the window size by a factor of $(Q+1)$, and decreasing the working set sizes by V. The following are the transformation rules:

(1) $t_i^* \leftarrow \lceil t_i/(Q+1) \rceil$;

(2) $w^*(t_i^*) \leftarrow w(t_i) - V$, if $w(t_i) > V$,
$w^*(t_i^*) \leftarrow 1$, otherwise;

(3) $f_j^* \leftarrow \lceil f_j/(Q+1) \rceil$;

(4) $T^* \leftarrow \lceil T/(Q+1) \rceil$.

Note that the length of r^* is only approximately $1/(Q+1)$ times the length of the reference string r'' produced by the execution of the synthetic program; also, r'' is expected to exhibit the given (w,f) characterization with a reasonably small error.

Clearly, the above transformation rules do not guarantee that (w^*,f^*) will be a valid characterization, i.e., that it will satisfy conditions (a) through (f) of Theorem 1. If any of those conditions is not satisfied, GENWS will not be able to generate r'', and the synthetic program will not be produced by the approach described in Fig. 10. The properties stated in Theorem 1 can be translated into the following three conditions, which the reduced (w,f) characterization must satisfy in order for (w^*,f^*) to be valid:

(A) $|w(t_j) - w(t_i)| \leq |t_j - t_i|/(Q+1)$ for any two points of coordinates $(t_i,w(t_i))$ and $(t_j,w(t_j))$ appearing in the reduced w string;

(B) $w_{max} - V \leq \lceil T/(Q+1) \rceil$;

(C) if $t_m - t_1 < T$, then $\Delta(t_1,t_m) < w(t_1) - V$, for any substring $w(t_1), w(t_2), \cdots w(t_m)$ of w, where

$$\Delta(t_1,t_m) = \sum_{\substack{i=1 \\ w(t_i)>w(t_{i+1})}}^{m-1} w(t_i) - w(t_{i+1}) + \phi(t_1,t_m)$$

is the minimum number of departures that must occur between t_1 and t_m, and $\phi(t_1,t_m)$ is the total number of flat faults occurring in (t_1,t_m).

Condition (A) is a consequence of applying condition (d) to w^*: for any t_i^* and t_j^*, we must have

$$|w^*(t_j^*) - w^*(t_i^*)| \leq |t_j^* - t_i^*|,$$

which, because of transformation rules (1) and (2), and assuming $w(t_i) > V$, $w(t_j) > V$, becomes condition (A). Note that, when either or both of the values of $w(t)$ being considered is not greater than V, condition (A) is sufficient to ensure the fulfillment of condition (d), though it is no longer necessary.

It is clear now why there should be, as stated in the previous section, some minimum interval of time between the points selected to represent the $w(t)$ curve: the length of this interval is based on the value of Q. For example, if Q=4 and V=2, the valid pair of input points with coordinates (20,5) and (25,8) would be transformed into the invalid pair (4,3) and (5,6). Thus, one consequence of the transformation is that slopes that are too steep, as defined by condition (A), will not be reproducible. Since most programs exhibit temporal locality, however, the longer the time interval between selected points, the greater the probability that condition (A) holds.

Condition (B) stems from property (b) of Theorem 1, and (C) comes from (f), but is slightly more restrictive, since the transformation reduces the working set size by V pages. For example, if Q=4, V=2, and T=50, then the valid reduced substring {(50, 5), (70, 8), (90, 4)} is transformed into the substring {(10, 3), (14, 6), (18, 2)}, which is invalid since the number of departures required (6 - 2 = 4) within the new window T^* = 10 exceeds the initial working set size ($w^*(t_1^*)$ = 3).

One other point to be noted is that the time interval between any two specified flat faults must be greater than Q.

A synthetic program was implemented according to the principles of BOOMERANG on a VAX 11/780 computer supporting a VMUNIX, a virtual memory version of Bell Laboratories' UNIX operating system developed at the University of California at Berkeley. The results we shall now present were obtained by performing trace-driven simulations of the execution of that program.

The actual implementation of a BOOMERANG program is very simple, and the design concepts can easily be kept machine-independent. In our implementation, we were able to limit the number of extraneous references to Q=8, and the number of distinct extraneous pages accessed to V=4. (Of the eight extraneous references, five access the page containing the BOOMERANG program's instructions, one accesses the page containing the current page address, and two access two other pages for index calculations.)

The eight extraneous references occur in a fixed sequence, and are all invariant except for the single reference to the page containing the current segment of r^*, which will change every time a segment of r^* has been entirely used. Transformation rule (2) is correct, or almost, if we assume that this reference remains invariant over a time interval whose length is on the order of T. (While the VAX 11/780 uses 512-byte pages, VMUNIX, the Berkeley-developed paging version of the Bell Laboratories' UNIX operating system, defines pages as 1024-byte blocks. Such a block could hold 512 2-byte page names, so one block is accessed to generate $512 \times (8 + 1) = 4608$ references of r'' before a new one is needed.)

The question to be investigated was whether and how accurately our implementation of BOOMERANG could reproduce a given (w,f) dynamics under a working set policy with realistic values of T. Our previous discussion indicated that the answer depends primarily on the value of Q and on the steepness of the slopes in the $w(t,T)$ curve to be reproduced.

Using the sampling methodology described in Section 4 to obtain an incomplete (w,f) characterization, experiments were performed with T=5,000, 10,000, and 20,000 references on the APL program trace introduced in the previous section.

For T=5,000, the transformed characterization was hopelessly invalid: too many points failed to satisfy condition (C). This was due to the large number of oscillations in the $w(t)$ curve for this value of T. Thus, at this relatively small window size, the approach was found to be unsatisfactory.

For T=10,000 (the near-optimal window size for this program), both conditions (B) and (C) were satisfied, and only for one pair of points was (A) not satisfied. This result seems to indicate that the "steep slopes" problem can almost be ignored for typical program behavior and reasonable window sizes. After adjusting one of the two points so that (A) would be satisfied over all the string, BOOMERANG could run, and the resulting $w(t,T)$ curve was, for all practical purposes, identical to that produced by GENWS. Similar conclusions were reached for T=20,000; again, the resulting curve was practically identical to that produced by GENWS.

The experiment with T=10,000 was repeated in a sampled working set environment following the scheme presented in Fig. 10, where, however, BOOMERANG was executed under the SWS policy, and the original (w,f) characterization was obtained by applying the sampling methodology described in Section 4 to the $w(t,T)$ curve generated by the SWS policy with a sampling interval equal to T. The (w^*,f^*) characterization corresponding to a specification of (w,f) based on only 434 points turned out to be a valid one. As shown in Table II, which displays the values of the main indices, and in Fig. 11, which compares the page faulting patterns of the two programs, BOOMERANG can provide a very accurate emulation of the dynamics of program memory demands.

6. Conclusions

The (w,f) characterization of a program's referencing dynamics presented in Section 2 has been shown to exhibit the following main properties:

(a) it lends itself easily to the construction of a generative deterministic model, and to the design and implementation of a synthetic program which is capable of reproducing a given dynamics with reasonable accuracy in working-set-like environments;

(b) while requiring substantially more information to be specified than most program behavior models, its accuracy is still quite acceptable with 3 or 4 orders of magnitude less information than a full reference string requires;

(c) it is based on variables having physical meanings, so that it is easy to modify it in a controlled way, to create hypothetical characterizations for the purposes of, say, testing the dynamic properties of a memory policy, and to study the relationships between it and the underlying program structure;

(d) it also allows generative stochastic models to be built, and to form the basis for synthetic programs with the characteristics summarized in property (a) above; these models will typically need considerably smaller amounts of information to be specified than those required by the deterministic model discussed in this paper.

The major drawback of the (w,f) characterization is its lack of robustness with respect to memory policies and policy parameters. Although it is intended to be a policy-independent characterization, in that it describes the time variations of a program's memory demand, the accuracy of an artificial string or of a synthetic program having a given dynamic characterization under the WS policy with a certain value of the window size T is sensitive to the value of T with which the string or the program is processed (if T is decreased), and even more to the memory policy under which it is processed. The problem of varying window sizes can probably be solved, or eased, by a characterization based on two or three working set size strings and flat fault strings corresponding to two or three different, suitably chosen values of T. This more complex characterization, expected to be more robust even with respect to variations in the type of memory policy, for which some changes to the algorithm described in Section 3 have also been proposed in Section 4, will be the subject of a forthcoming paper.

Acknowledgements

The authors are grateful to the members of the PROGRES group for the many discussions on the topics of this paper, to T. Paul Lee, who performed the experiments in Section 4b, and to Susan Hanson for her patient and skillful typing of the manuscript.

References

[1] T. Alanko, I. Aikala and P. Kutvonen, Methodology and empirical results of program behavior measurements, Proc. Performance 80, the 7th Int. Symp. on Computer Performance Modelling, Measurement and Evaluation (May 1980), 55-66.

[2] O. Babaoglu, On Constructing Synthetic Programs for Virtual Memory Environments, in: D. Ferrari and M. Spadoni, eds., Experimental Computer Performance Evaluation (North-Holland Publ. Comp., Amsterdam, 1981), 195-204.

[3] P. Bryant, Predicting working set sizes, IBM J. Res. Develop. 19 (1975), 221-229.

[4] W.W. Chu and H. Opderbeck, Program behavior and the page fault frequency replacement algorithm, Computer 9 (Nov. 1976), 29-38.

[5] E.J. Coffman, Jr., and P.J. Denning, Operating Systems Theory (Prentice-Hall, Englewood Cliffs, 1973).

[6] P. J. Denning and K.C. Kahn, A study of program locality and lifetime functions, Proc. 5th SIGOPS Symp. Operating Systems Principles (Nov. 1975), 207-216.

[7] P.J. Denning and S.C. Schwartz, Properties of the working set model, Comm. ACM 15 (1972), 191-198.

[8] N. Faller, Design and analysis of program behavior models for the reproduction of working-set characteristics, Ph.D. dissertation; also, Memo No. UCB/ERL M81/59, PROGRES Report No. 81.7, University of California, Berkeley (June, 1981).

[9] D. Ferrari, A generative model of working set dynamics, Proc. SIGMETRICS Conf. on Measurement and Modeling of Computer Systems (Sept. 1981), 52-57.

[10] D. Ferrari, Characterization and Reproduction of the Referencing Dynamics of Programs, in: F. Kylstra, ed., Performance 81 (North-Holland Publ. Comp., Amsterdam, 1982), 363-372.

[11] A.J. Smith, A modified working set paging algorithm, IEEE Trans. Computers C-25 (1976), 907-914.

[12] J. Spirn, Program Behavior: Models and Measurements (Elsevier North-Holland, New York, 1977).

Table I
Experimental results under WS with T=10,000

Index		R	R'	Error
Mean working set size	$\overline{w}$	20.90	20.92	+0.1%
Maximum working set size	w_{max}	56	56	0
Space-time product	$stp \times 10^{-8}$	1.067	1.058	-0.8%
Page fault rate	$\overline{f} \times 10^{4}$	6.48	6.44	-0.6%
Mean absolute difference between working set sizes	$\overline{\delta}$	0.228		1.0%

Table II
Experimental comparison between the APL program and its reproduction by the BOOMERANG synthetic program
(policy: SWS; T=10,000)

Index		APL	BOOMERANG	ERROR
Mean working set size	$\overline{w}$	23.82	23.98	+0.6%
Maximum working set size	w_{max}	78	77	-1.2%
Space-time product	$stp \times 10^{-8}$	1.070	1.049	-1.9%
Page fault rate	$\overline{f} \times 10^{4}$	5.82	5.72	-1.7%
Mean absolute difference between working set sizes	$\overline{\delta}$	.057		2.39%

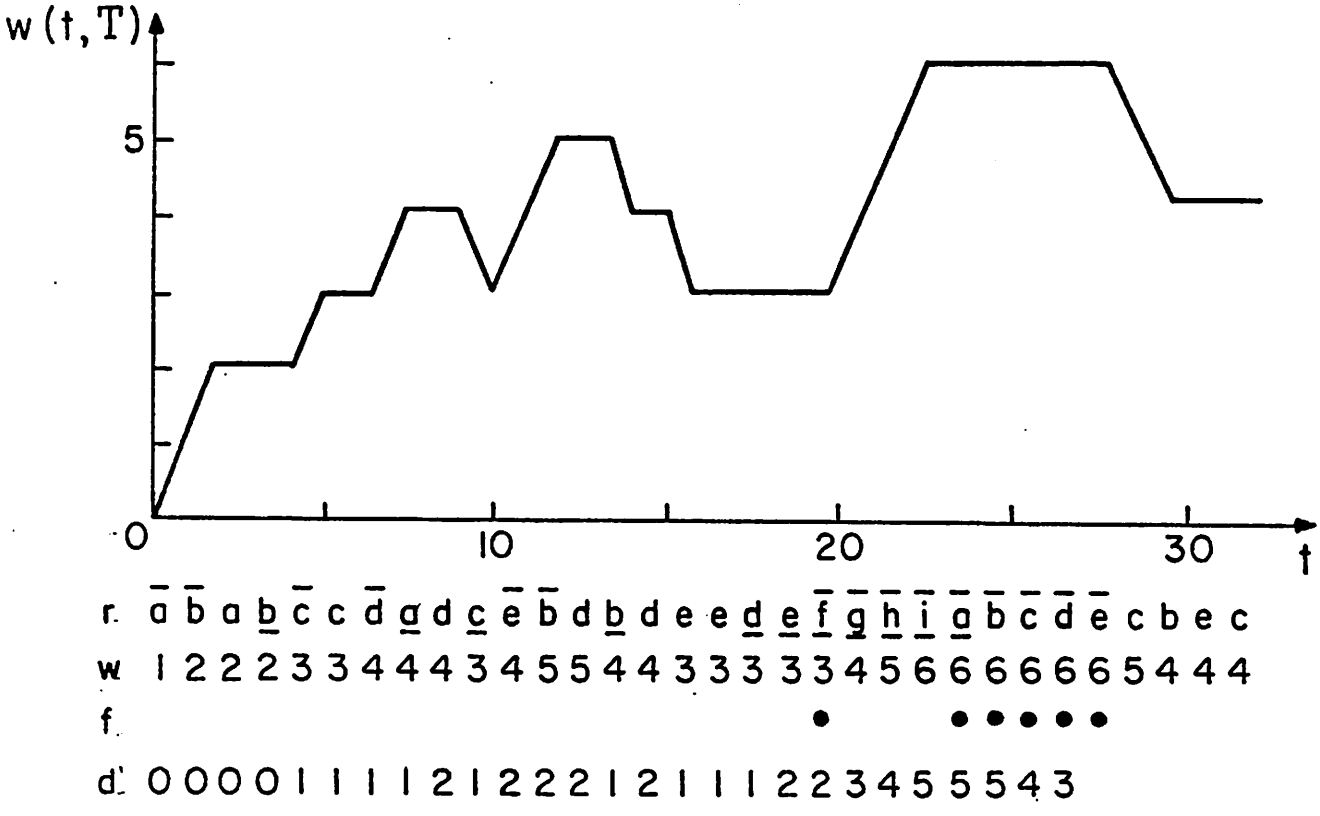

Fig. 1 A reference string r, its working set size string w for $T=6$, its flat fault string f (whose entries are the times marked by a dot), its $w(t,T)$ curve, and its decrement string d. Note that the references with a bar correspond to page arrivals, and the underlined ones to page departures.

Inputs: $n, k, T, P, w_t \ (t = 1,n), f_j \ (j = 1,k)$.
Output: $r_t (t = 1,n)$.
Initialization
1. $C \leftarrow \emptyset$; $F \leftarrow \emptyset$; $E \leftarrow P$; $w_0 \leftarrow 0$; $f_{k+1} \leftarrow 0$; $t \leftarrow 1$; $j \leftarrow 1$;
2. for $i \leftarrow 1$, T do $w_{n+i} \leftarrow w_n$;
Reference Selection
3. if $w_t > w_{t-1}$ or $f_j = t$ then begin if $E = \emptyset$ then error; $r_t \leftarrow e \ \varepsilon \ E$; $E \leftarrow E - \{e\}$ end
   else begin if $C = \emptyset$ then error; $r_t \leftarrow first(C)$; $C \leftarrow C - first(C)$ end;
4. if timeindex $[first(C)] = t$ then error;
Set Updating
5. if $w_{t+T} < w_{t+T-1}$ or $f_m = t + T$ for some $m \geq j$ then $F \leftarrow F \cup \{r_t[t + T]\}$
   else $C \leftarrow C \cup \{r_t[t + T]\}$;
6. if timeindex $[first(F)] = t$ then begin $E \leftarrow E \cup \{first(F)\}$; $F \leftarrow F - \{first(F)\}$ end;

Loop Control
7. if $f_j = t$ then $j \leftarrow j + 1$;
8. if $t < n$ then begin $t \leftarrow t + 1$, go to 3 end else stop.

Fig. 2. A description of the string generation algorithm.

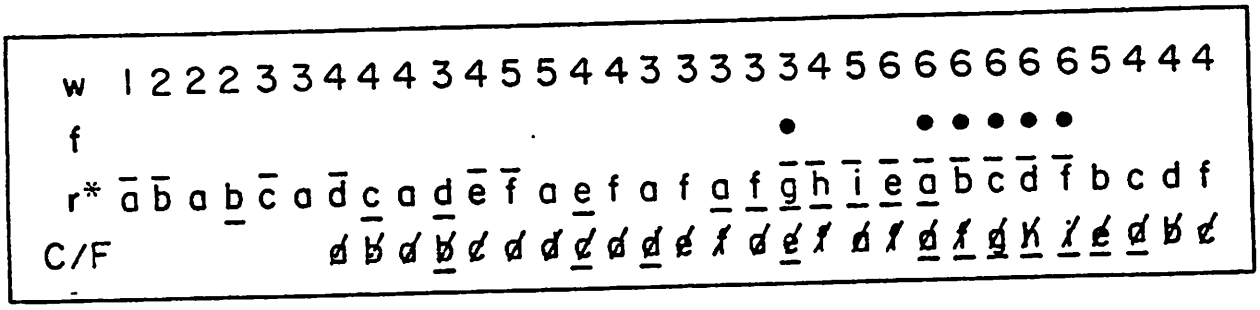

Fig. 3. Generation of a string r^* with the same (w,f) characterization as string r in Fig. 1 (T=6).

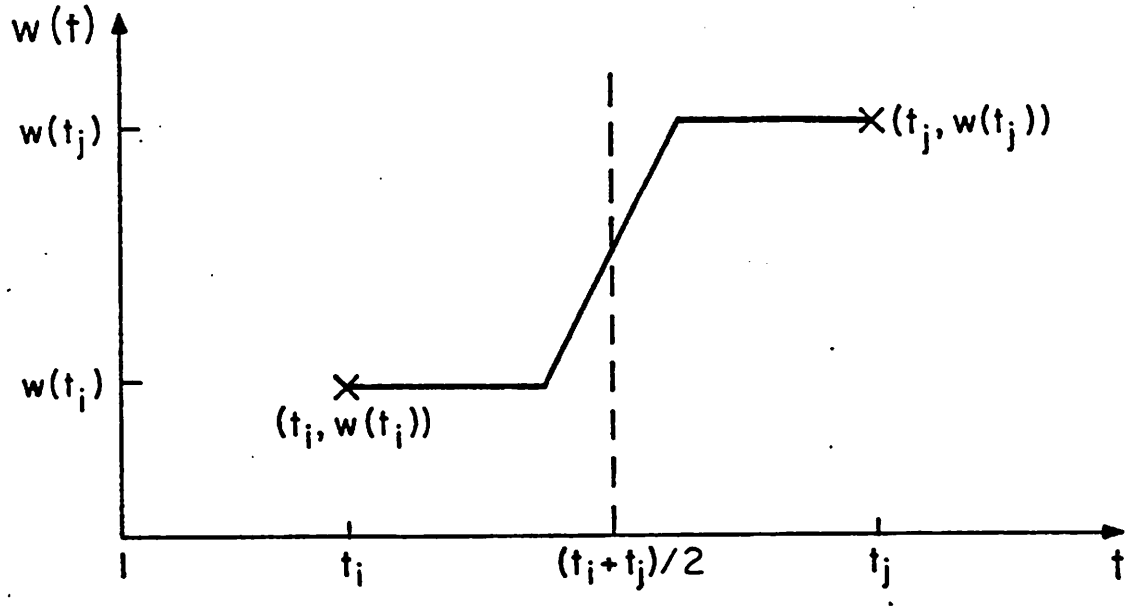

Fig. 4. The interpolation technique for $\dot{w}(t_j) > w(t_i)$.

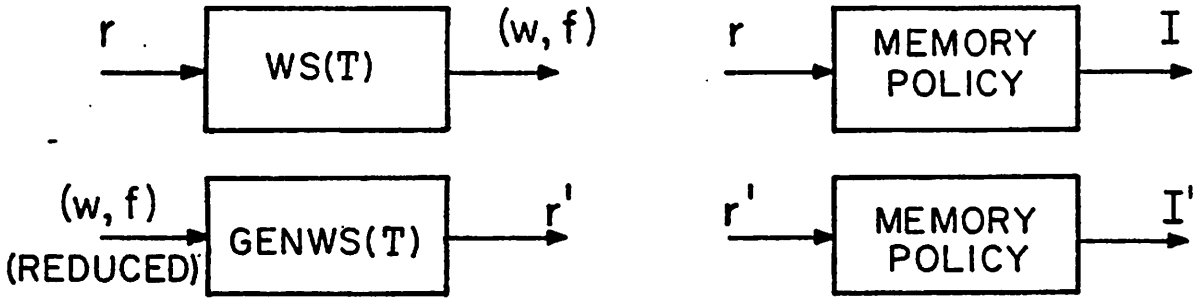

Fig. 5. Block diagram of the model validation experiments.

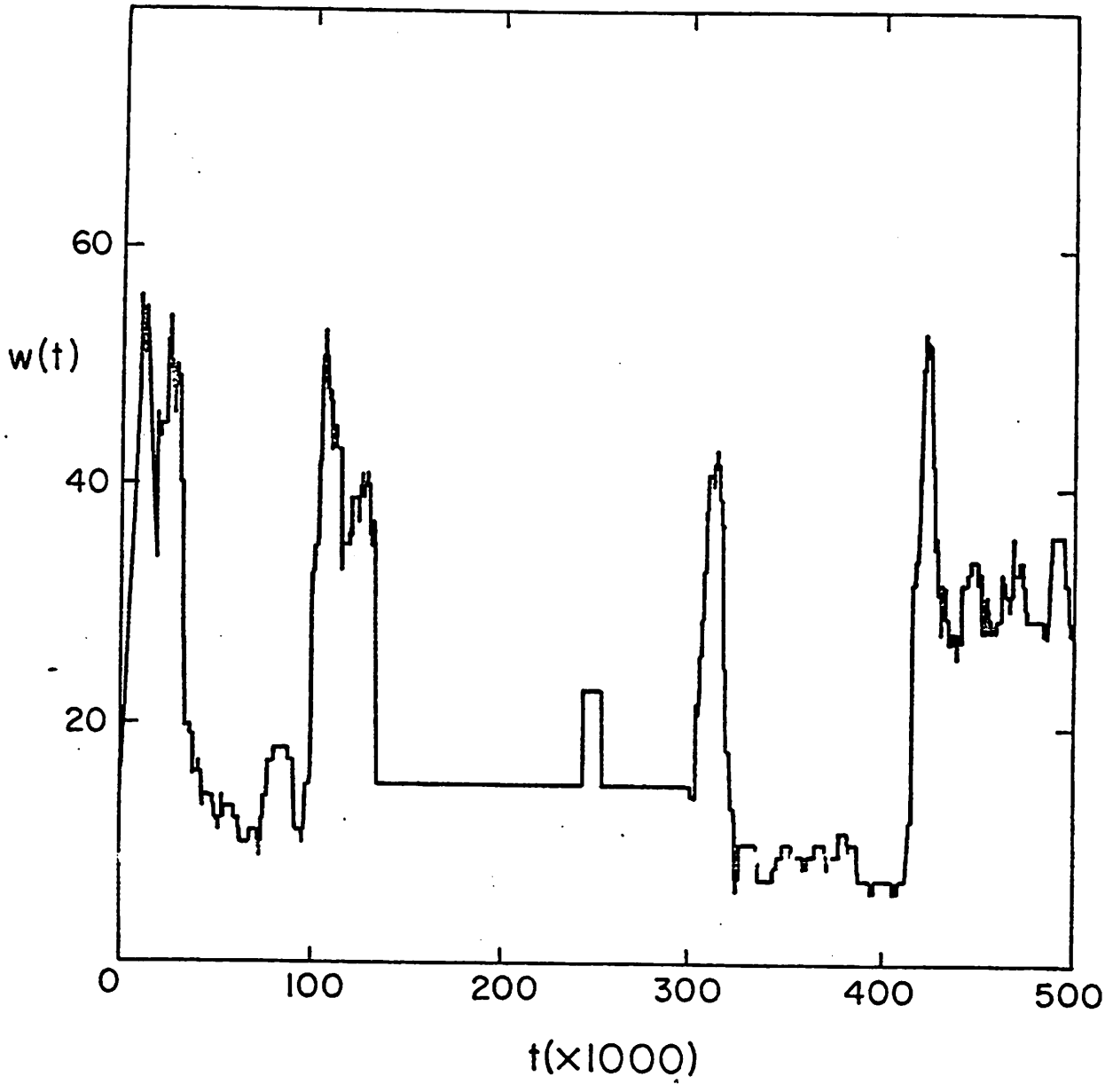

Fig. 6. The $w(t)$ curve for the APL trace with $T=10{,}000$ references.

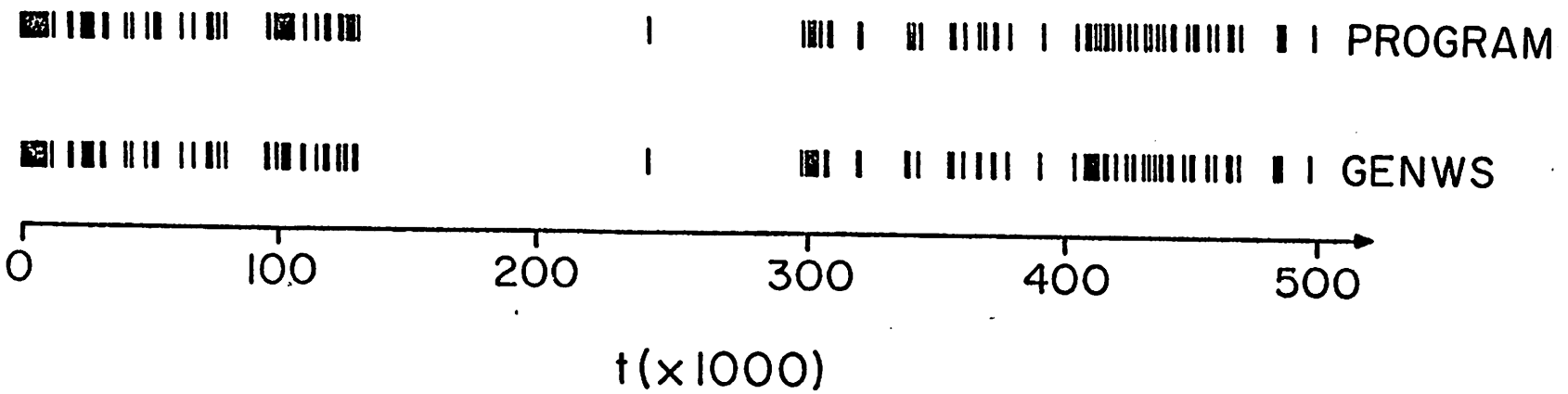

Fig. 7. Page fault patterns generated by strings r (PROGRAM) and r' (GENWS) under a WS policy with $T=10,000$ references.

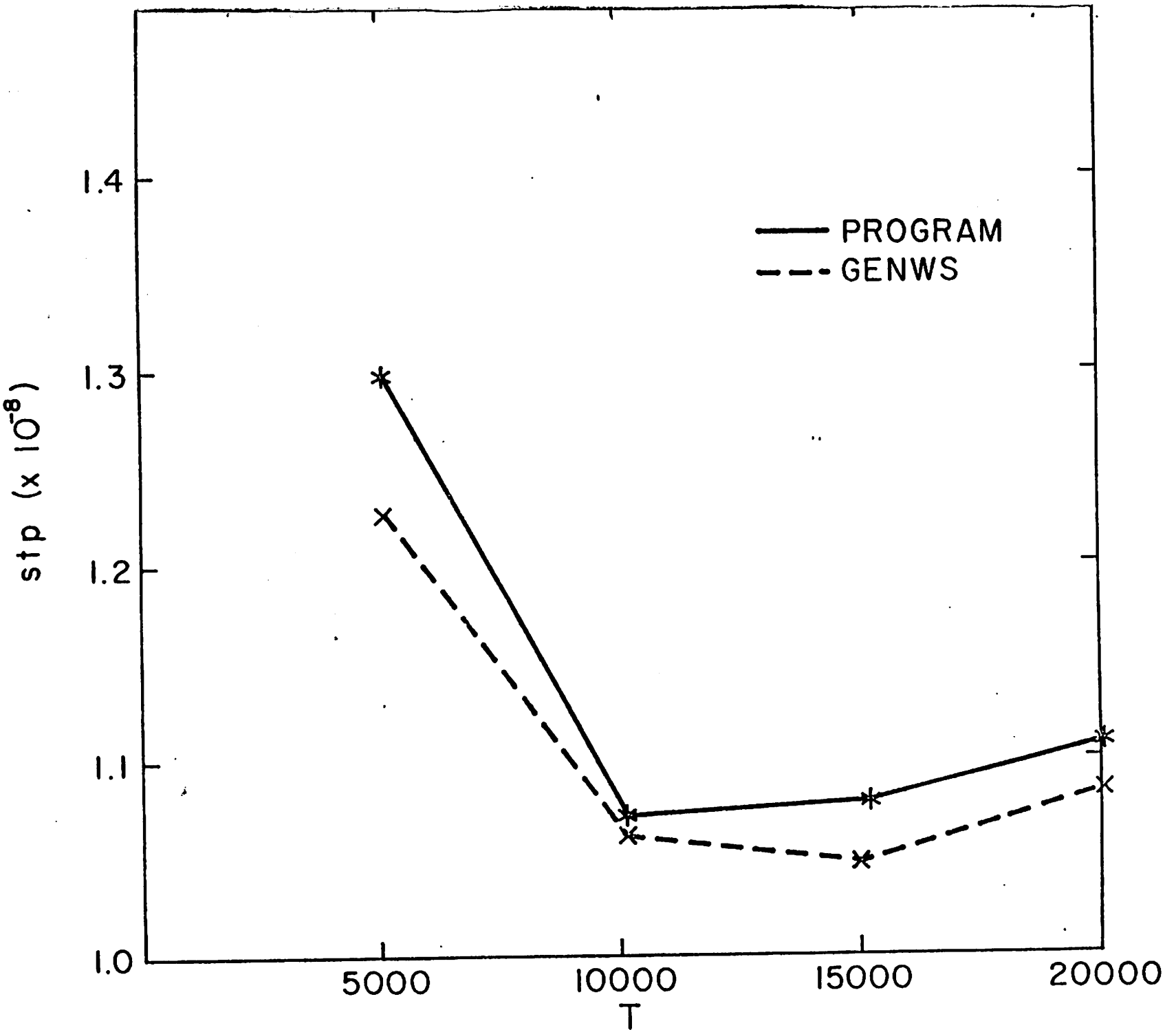

Fig. 8. Space-time product vs. the window size T used in the (w,f) characterization for strings r (solid line) and r' (broken line).

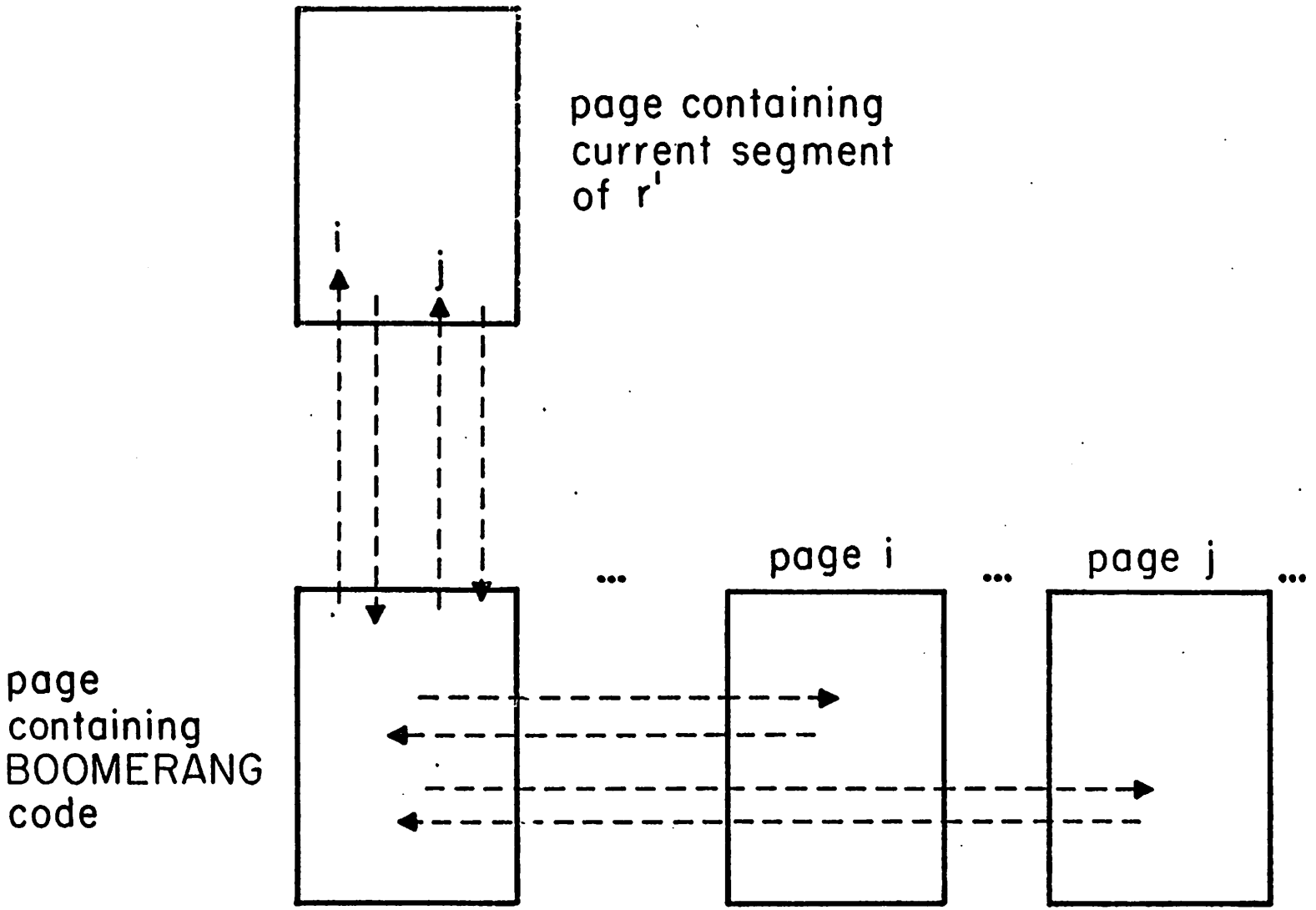

Fig. 9 Operation of the BOOMERANG program.

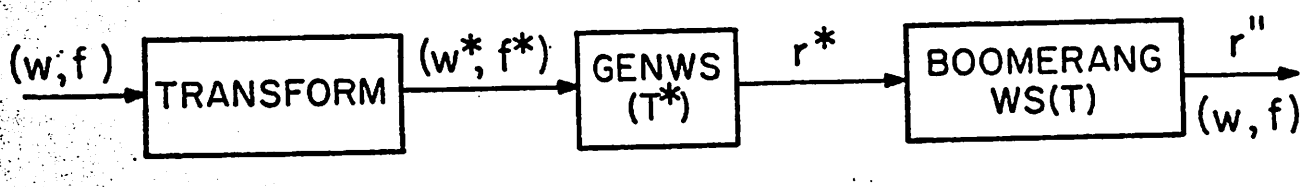

Fig. 10. Transforming a (w,f) characterization to compensate for the extraneous references produced by the BOOMERANG synthetic program.

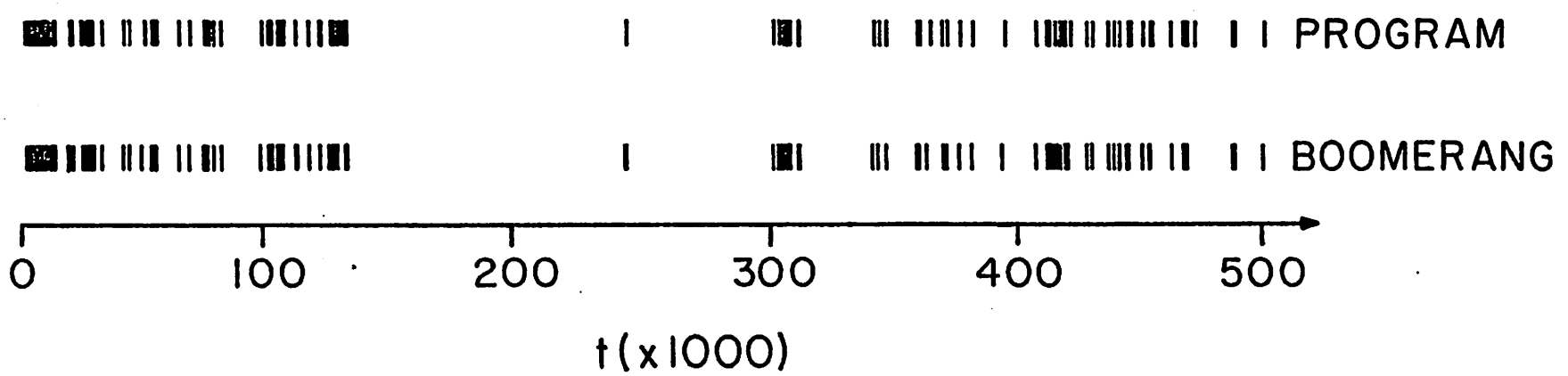

Fig. 11. Page fault patterns generated by the APL program (PROGRAM) and by its BOOMERANG synthetic reproduction, under the SWS policy with T=10,000 references.