EFFICIENT GENERATION OF MEMORY REFERENCE STRINGS BASED ON THE LRU STACK MODEL OF PROGRAM BEHAVIOUR*

Özalp Babaoğlu

TR 81-475
October 1981

Department of Computer Science
Cornell University
Ithaca, New York 14853

* Partial support for this work was provided by the National Science Foundation under Grants No. MCS 7824618 and MCS 7807291.

Efficient Generation of Memory Reference Strings Based on the LRU Stack Model of Program Behaviour†

Özalp Babaoğlu

We consider the problem of generating memory reference strings that are to be used instead of real program address traces with the generator based on the Least-Recently-Used Stack Model (LRUSM) of program behaviour. A method to transform the stack distance probability mass function that drives the generator is proposed which results in memory reference strings that are a fraction of the original string length while preserving most of its essential performance characteristics. The reduced string can be processed in much the same way as the original string for virtual memory studies that deal with memory sizes greater than k, the parameter of the transformation.

1. Introduction

Trace-driven simulation is a frequently used method for studying the performance of various aspects of storage hierarchies ranging from

† Partial support for this work was provided by the National Science Foundation under Grants No. MCS 7824618 and MCS 7807291

cache buffers to file systems [Bel66a, Lau79a, Mat70a, Smi81a]. Although we will limit our discussion to the classical primary memory-secondary memory level of the hierarchy in a paging environment, the ideas and results are naturally applicable to the others levels. The trace data used in these studies consists of a record of all the memory accesses (data and instruction) generated during an interval in the execution of a program. One of the drawbacks of this approach is the fact that the selected programs have to be interpretively executed in order to obtain the desired address traces. Furthermore, simulation studies dealing with realistically long trace data (a few million references) are very costly both in space and time. To reduce these costs, most trace-driven simulation studies work with a _reduced_ version of the original trace. A scheme to obtain such a reduced trace was first proposed by Coffman and Randell [Cof71a] for studies employing the class of _stack policies_ [Mat70a]. The method was recently extended to _working set_ environments by Alanko et al [Ala80a]. In both schemes, given a control parameter, the reduced trace is constructed by recording the events corresponding to the _entry_ and _exit_ of pages from the set of memory resident pages as the original trace is processed by the particular management policy with the given control parameter. Due to the inclusion property of stack policies [Mat70a] and of the working set policy [Fra78a] with respect to their parameters, studies employing values of policy parameters greater than those used for the reduction can be carried out on the reduced trace with exact results.

Smith [Smi77a] has studied two other methods for reducing existing trace data that are approximate in nature. The first of his methods, the _Stack Deletion Method_ with parameter k, removes from the original trace data all references to pages that are elements of the set of k _most recently used_ pages. In other words, this scheme is identical to the one proposed by Coffman and Randell for the _Least Recently Used_ (LRU) policy aside from the fact that the _exit_ events corresponding to pages leaving the top k elements of the stack are omitted in the reduced trace. The second method studied by Smith, the _Snapshot_ or _Reference Set_ method [Pri74a, Lau79a] with sample interval T removes from the original trace data all references that are re-references to pages within a given sample interval of length T. The claim that these compression techniques preserve the essential performance characteristics (namely, the page fault rate and the mean memory occupancy) of the original reference string has been verified empirically when they are processed by a wide variety of paging algorithms [Smi77a].

An alternative approach to reducing the space cost of such studies is to use a model of program behaviour such as the _Independent Reference Model_ (IRM) [Aho71a, Bas76a, Spi72a] or the _Least-Recently-Used Stack Model_ (LRUSM) [Rau77a, Spi76a] in conjunction with a random number source, thus obtaining a generative model [Spi77a]. Since most such models require a small fixed number (usually proportional to n, the number of pages contained in the program) of parameters to identify them, memory reference strings of arbitrary length can be generated one reference at a time with essentially no space cost. Note that, from the operational viewpoint of the simulator, the trace-driven and the generator-driven methods are identical.

In addition to their ability to generate reference strings of arbitrary length, probabilistic models of program memory reference behaviour are compact, often analytically tractable, and can be modified to represent diverse behaviours through an appropriate perturbation of their parameters [Spi76a]. The goal of this paper, however, is not to advocate the global substitution of actual address traces with probabilistic models. We are simply interested in an efficient method for generating through the LRUSM memory reference strings to be used in studies where the predictive capabilities of this particular model are deemed to be of sufficient accuracy. Further discussion of the nature of these studies, along with a formal description of the LRUSM, follows in the next section. Section 3 defines the transformation method and investigates its implications on the string length, the steady-state fault rate, and the mean memory occupancy. Section 4 is a comparison of our proposed transformation method with the Stack Deletion Method. Finally, in section 5 we discuss a novel application of the developed method to the design of synthetic programs that are to exercise virtual memory systems.

2. The LRU Stack Model

2.1. The Descriptive Model

The model of program behaviour that our study will be based upon is the LRUSM . In this model, the n pages that constitute the program's virtual address space are envisioned to be ordered in a stack according to their recency of usage. The page referenced at discrete time t, r_t, is the one currently occupying stack position d_t, where we assume that the page references occur at equidistant time points and the interval between references is taken to be the unit of time. The sequence of stack positions $\{d_i\}$ is called the distance string. The LRUSM assumes the d_i's to be independent and identically distributed samples from the population (1,2,3,...,n) with the stationary probabilities $Pr\{d_t=j\}=\delta_j$. The probability mass function (pmf) $D=(\delta_1, \delta_2, \ldots, \delta_n)$, where $\sum_{i=1}^{n} \delta_i=1$ and $\delta_i>0$ for all i, uniquely defines the LRUSM. An instance of the LRUSM corresponding to a particular program execution is realized upon providing point estimates for the n parameters of D †. Each generated reference causes the stack to be updated by placing the referenced page, r_t, at the top (stack position 1) and all pages in stack positions 1 through d_t-1 to be shifted down one position so as to preserve the LRU ordering [Mat70a]. The set of pages occupying stack positions below d_t remain unaffected. In terms of this description of the LRUSM, the Stack Deletion Method of trace data compression with parameter k can be characterized as removing from a given trace all references that are to stack positions less than or equal to k.

Properties of the LRUSM based on analytic and experimental studies have been extensively reported in the literature [Lew71a, Lew73a, Rau77a, Smi76a, Spi76a]. Although it is able to model the steady-state page fault rate and the mean memory occupancy characteristics of real programs under a variety of paging algorithms with reasonable accuracy [Raf76a, Smi76a, Spi72a] (prediction errors in one study [Spi72a] averaged about 10% for working set size and about 30% for page fault rate), the LRUSM has several known shortcomings at a more microscopic level of program behaviour. In particular, the assumptions of independent and identically distributed stack distances in the LRUSM

† Actually, only (n-1) of these parameters are independent.

have been formally shown to be inadequate for particular address traces [Lew73a, Lew71a]. The inter-fault times which are independent, identically and geometrically distributed with a constant parameter as a consequence of this lack of serial correlation in the distance string for the LRUSM, have been empirically observed to be correlated and to have highly skewed distributions with long tails [Lew73a, Lew71a]. Finally, the property of the reference string generated by the LRUSM that results in uniform access to all pages of the program is highly unrepresentative of real programs [Bas76a, Lau79a, Cof73a]. In light of these shortcomings, the use of the LRUSM as a *micromodel* in a two-level model of program behaviour is more appropriate [Den80a]. As an example, the second level, or *macromodel*, could represent the phase transitions and consist of a semi-Markov chain [Kah76a].

2.2. The Generative Model

Constructing a generative model based on the LRUSM simply involves generating the distance string $\{d_i\}$ as independent samples from the population $(1,2,3,\ldots,n)$ according to D (the stack distance pmf D is said to *drive* the generator) and obtaining the reference string $\{r_i\}$ through the required stack manipulations. The d_i's can be obtained by transforming a sequence of uniform pseudo random numbers through a technique such as the *aliasing* scheme of Walker [Wal77a].

One possible approach to reducing the time cost of a generator-driven simulator is the following. The generator is allowed to run unaltered as described above. The resulting reference string is processed by the Stack Deletion Method of Smith with parameter k, thus resulting in a compressed version of the generated string. This compressed string is then used as the actual input to the simulation study. What we seek is a method for combining the generation and compression functions whereby we obtain the compressed string directly as the output of the generation process.

2.3. Model Use

In light of the deficiencies listed in section 2.1., the use of the LRUSM in a particular study has to be justified; no matter how efficient the generation process. These accuracy and validity concerns are universal to all modeling efforts and must be resolved prior to model use. Typically, the selected predictions of an instance of the model are compared to measured results and are judged to be of desired accuracy. This step is repeated for several other instances of the model to establish a domain of validity. Model use can then proceed and obtain further predictions from yet different model instances with attention paid to keeping it within the above domain [Spi77a].

Based on published results regarding the accuracy of the LRUSM as a predictor of the page fault rate and mean memory occupancy [Raf76a, Smi76a, Spi72a] , its use is natural for studies interested in these measures where real program trace data is unavailable. It is also conceivable to use LRUSM generated traces to develop and test simulators that will eventually run using real trace data in an effort to minimize the cost of program development. Perhaps the most common application of model generated traces, however, is found in those studies interested in the relative rather than absolute performances of various management schemes and/or algorithms and data structures used to process trace data. As an example, note the study by Olken [Olk80a] comparing three different data structures for obtaining LRU fault rate statistics for extremely large disk caches where the file access string is generated through the LRUSM. Finally, for certain measures of interest in a given environment, the LRUSM becomes analytically untractable. In such cases one naturally resorts to simulation based on the LRUSM. The analysis by Rau [Rau79a] of the effective bandwidth of an interleaved memory system, where the program memory module reference behaviour is modeled through the LRUSM, is a case in point.

3. The Transformation Method

In this section we will develop a method (called the transformation method) for modifying the stack distance pmf of an LRUSM, and will formally show that the resulting string preserves the steady-state page fault rate of the original string while being only a fraction of its length. In section 3.2., the implications of this transformation on the mean memory occupancy statistic are investigated.

3.1. Fault Rate Characteristics

Our strategy in developing the method will be as follows. We will review some of the basic properties of the LRUSM and eventually derive an expression of a confidence interval for the steady-state page fault rate. We will repeat these steps also for an incompletely specified LRUSM obtained from the original through a transformation of the stack distance pmf. The requirement that a statistic based on this new LRUSM result in a confidence interval of the same width at the same level as the original LRUSM will then be used to completely define the transformation.

Let $D_1 = (\delta_1, \delta_2, \ldots, \delta_n)$ be the driving stack distance pmf for the original generator, hereafter referred to as GEN1. For D_1, we define the cumulative probabilities as $\Delta_j = \sum_{i=1}^{j} \delta_i$.
The events corresponding to page faults for the page reference string generated by GEN1 in a fixed memory partition of m pages managed by the LRU policy constitute a discrete time renewal process. In particular, the inter-fault time distribution is geometric with parameter $\lambda_m = \sum_{i=m+1}^{n} \delta_i = (1-\Delta_m)$ [Spi76a].

Let the random variable $N_1(t_1;m)$ denote the number of faults generated until some arbitrary time t_1 in a fixed size memory of m pages. By renewal theory [Ros70a], the steady-state fault rate is given by

$$\lim_{t_1 \to \infty} \frac{N_1(t_1;m)}{t_1} = (1-\Delta_m) = \lambda_m . \tag{3.1.1}$$

Equation (3.1.1) holds with probability 1.

For $t_1 < \infty$, however, applying Chebyshev's inequality for the random variable $N_1(t_1;m)/t_1$ we have

$$\Pr\left[\left| \frac{N_1(t_1;m)}{t_1} - \lambda_m \right| < \epsilon \right] > 1 - \frac{\sigma^2(t_1)}{\epsilon^2} \tag{3.1.2}$$

for any $\epsilon > 0$ and where $\sigma^2(t_1) = \mathrm{Var}(N_1(t_1;m)/t_1)$.

Since, for fixed t_1 and m, the random variable $N_1(t_1;m)$ has a binomial distribution with parameters $(1-\Delta_m)$ and t_1, the variance $\sigma^2(t_1)$ is given by

$$\sigma^2(t_1) = \mathrm{Var}(N_1(t_1;m)/t_1) = (\Delta_m(1-\Delta_m))/t_1. \qquad (3.1.3)$$

Note that $\left[\frac{N_1(t_1;m)}{t_1} - \epsilon , \frac{N_1(t_1;m)}{t_1} + \epsilon \right]$ is a level $(1 - \frac{\sigma^2(t_1)}{\epsilon^2})$ confidence interval of length 2ϵ for λ_m.

Now consider a second generator, called GEN2, that is driven by the following transformed stack distance pmf:

$$D_2 = (\delta'_1, \delta'_2, \ldots, \delta'_k, R\delta_{k+1}, R\delta_{k+2}, \ldots, R\delta_n)$$

where k is the fixed parameter of the transformation and R is an arbitrary positive constant.

Note that the transformation of δ_i $(1 \le i \le k)$ is left unspecified; the only constraint it has to satisfy is

$$\sum_{i=1}^{k} \delta'_i = 1-R(1-\Delta_k) \qquad (3.1.4)$$

so that D_2 is indeed a pmf.

As before, let the random variable $N_2(t_2;m)$ denote the number of faults resulting from GEN2 by some arbitrary time t_2 under the same circumstances as with GEN1 (i.e., fixed memory partition of m pages managed by the LRU policy).

Note that, for GEN2 with $k<m$, the statistic $N_2(t_2;m)/Rt_2$ is an unbiased estimate of λ_m. Furthermore,

$$\mathrm{Var}(N_2(t_2;m)/Rt_2) = (1-\Delta_m)(1-R(1-\Delta_m))/Rt_2. \tag{3.1.5}$$

since $N_2(t_2;m)$ is binomially distributed with parameters $R(1-\Delta_m)$ and t_2. As for GEN1, the level of a confidence interval of length 2ϵ for λ_m in terms of this new statistic is given by

$$\Pr\left[\left|\frac{N_2(t_2;m)}{Rt_2} - \lambda_m\right| < \epsilon\right] > 1-\frac{\mathrm{Var}(N_2(t_2;m)/Rt_2)}{\epsilon^2}. \tag{3.1.6}$$

The motivation for the transformation method is evident in the following observation: for some given t_1, there exist values of R and t_2 with $t_2<t_1$ such that $N_2(t_2;m)/Rt_2$ is as good a statistic for λ_m as $N_1(t_1;m)/t_1$. We formalize this statement in the following proposition:

Proposition : For some given memory size m, time t_1, and transformation parameter $k<m$, the smallest possible time t_2 for which the statistic $N_2(t_2;m)/Rt_2$ achieves the same confidence as $N_1(t_1;m)/t_1$ for the steady-state fault rate λ_m is obtained by a D_2 where $R=R^* = 1/(1-\Delta_k)$.

Proof : Equating the right hand sides of equations (3.1.2) and (3.1.6) corresponding to the confidence levels due to GEN1 and GEN2 respectively for a given ϵ, and substituting equations (3.1.3) and (3.1.5) for the two variances, we obtain

$$1-\frac{\Delta_m(1-\Delta_m)}{\epsilon^2 t_1} = 1-\frac{(1-\Delta_m)(1-R(1-\Delta_m))}{\epsilon^2 Rt_2}. \tag{3.1.7}$$

Simplifying and solving for t_2 yields

$$t_2 = \frac{t_1(1-R(1-\Delta_m))}{R\Delta_m}. \qquad (3.1.8)$$

Thus, the minimization of t_2 as a function of R can be formulated as the following optimization problem:

$$\text{minimize } t_2 = t_1(1-R(1-\Delta_m))/R\Delta_m$$
$$\text{subject to } \quad 0 \le R \le \frac{1}{1-\Delta_k}$$

where the constraints simply ensure that D_2 is a valid pmf. Note that for given t_1 and m, t_2 is a strictly decreasing function of positive R. Therefore, the minimum t_2 value is that imposed by the upper bound of R. That is, the optimum R value is given by $R = R^* = \frac{1}{1-\Delta_k}$

$\square$

Evaluating t_2 at $R = R^*$ yields

$$t_2^* = \left(\frac{\Delta_m - \Delta_k}{\Delta_m}\right)t_1.$$

Note that

$$t_2^* \le \frac{t_1}{R^*}$$

with the equality holding when m=n. This observation leads us to the conclusion that the length of the string generated by GEN2 need only be at most $1/R^*$th the length of that due to GEN1 to achieve the same confidence for λ_m for all memory sizes greater than k.

Recalling the form of D_2 and equation (3.1.4), the stack distance pmf transformation implied by the optimal value of R is of the form

$$D_2 = (0,0,\ldots,0,R^*\delta_{k+1},R^*\delta_{k+2}, \ . \ . \ . \ , R^*\delta_n).$$

Given D_2, the operation of GEN2 is identical to that of GEN1 except that now the d_i's are generated as independent random variables with the frequencies specified by D_2.

We make the following observations about the transformation method:

(1) The transformation preserves the long-run relative occurrences of stack depths greater than k, i.e., $\delta_i/\delta_j = \delta'_i/\delta'_j$ for all i and j>k.

(2) GEN2 produces no references to stack depths less than or equal to k, while it references depths greater than k with increased probabilities. This result shows the analogy between the above scheme of generating memory references and the Stack Deletion Method of compressing existing memory reference trace data.

(3) The optimal value of R, R^*, may be interpreted as the expected number of references until the first reference to a stack depth greater than k. This confirms our earlier observation that GEN2 suppresses the references to the top k pages of the stack.

3.2. Mean Memory Occupancy Characteristics

In the previous section, we have shown that the proposed method preserves the page fault rate characteristics of the original string in an environment managed by a particular fixed partition policy (namely LRU). Due to its strong interaction with the process scheduling mechanism and significant impact on overall system performance, mean memory occupancy in a variable partition environment is another important property associated with a reference string. As an example of a variable partition policy, we will consider the behaviour of the output of GEN2 when processed by the working set algorithm [Den68a].

Let $\omega(\tau)$ denote the steady-state working set size with parameter (window size) τ. Recalling the definitions of the LRUSM and of the working set policy, the steady-state working set size distribution can be expressed through the recursive relationship

$$\Pr\{\omega(\tau)=i\} = \Delta_i \Pr\{\omega(\tau-1)=i\} + (1-\Delta_{i-1})\Pr\{\omega(\tau-1)=i-1\} \quad (3.2.1)$$

where $\Pr\{\omega(1)=1\} = 1$ [Spi77a].

As applied to the LRUSM that drives GEN2, equation (3.2.1) becomes

$$\Pr\{\omega(\tau')=i\} = \Delta'_i \Pr\{\omega(\tau'-1)=i\} + (1-\Delta'_{i-1})\Pr\{\omega(\tau'-1)=i-1\} \quad (3.2.2)$$

where $\Delta'_i = (\Delta_i-\Delta_k)/(1-\Delta_k)$ and the window is scaled down by a factor of R^*, i.e., $\tau' = (1-\Delta_k)\tau = \tau/R^*$, since each reference generated by GEN2 advances the clock by R^* ticks rather than by one†. If the δ_i's are nonzero for all $i>k$, equation (3.2.2), which is valid only for $i>k$, has

† Note that, as we have defined it, R^* being the mean of a random variable, is a real number. Conceptually, there is no reason why the clock of the simulator cannot be advanced by this nonintegral amount. However, for studies that require R^* to be integer, we can define $R^* = \lfloor 1/(1-\Delta_k) \rfloor$ (truncate it to an integer) in which case the probability measure $1-R^*(1-\Delta_k)$ as given by equation (3.1.4) will have to be assigned (in an arbitrary manner) to the first k stack positions.

the closed form solution [Spi77a]

$$\Pr\{\omega(\tau')=i\} = \sum_{j=k+1}^{i} \left[\frac{(1-\Delta'_{k+1})(1-\Delta'_{k+2})\cdots(1-\Delta'_{i-1})}{\prod_{\substack{l=k+1 \\ l\neq j}}^{i} (\Delta'_j-\Delta'_l)} \Delta'^{\,\tau'-1}_j \right], \quad k<i\leq n \qquad (3.2.3)$$

$$= 0, \quad i\leq k.$$

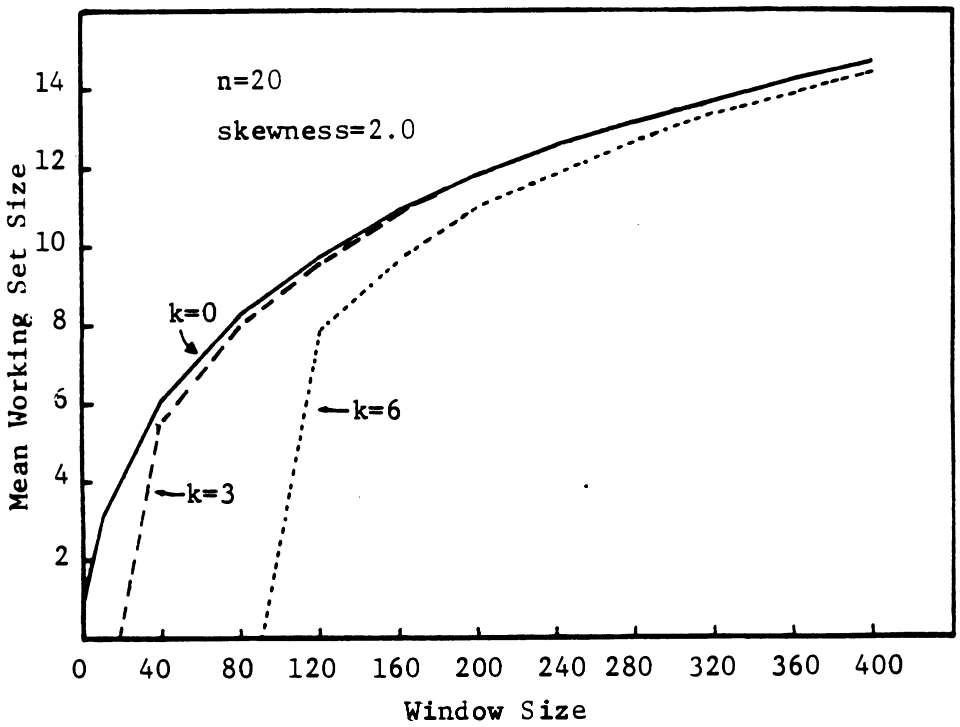

Fig. 3.2.1. Mean working set size

Having the distribution of $\omega(\tau)$ at hand, the mean, $\bar{\omega}(\tau)$, can be obtained trivially.

Ideally, one would like to show analytically that, for small k and large τ, the first moment of the above distribution approaches the mean working set size of the original string. However, due to the complexity of the expressions involved, only numerical results have been obtained.

Figure 3.2.1 presents the mean working set size of a sample program whose LRUSM parameters were obtained through Zipf's Law with skewness 2.0† [Knu73a]. The figure shows that the transformation preserves with good approximation the first moment of the working set size distribution even for large k, particularly when the window size τ is large. Recall that setting k=0 results in the null transformation and reduces GEN2 to GEN1. For the example at hand, the relatively large errors encountered for small values of τ are simply due to the scaling that is performed on the window size (which happens to be $\tau'=\tau/15.15$ when k=6).

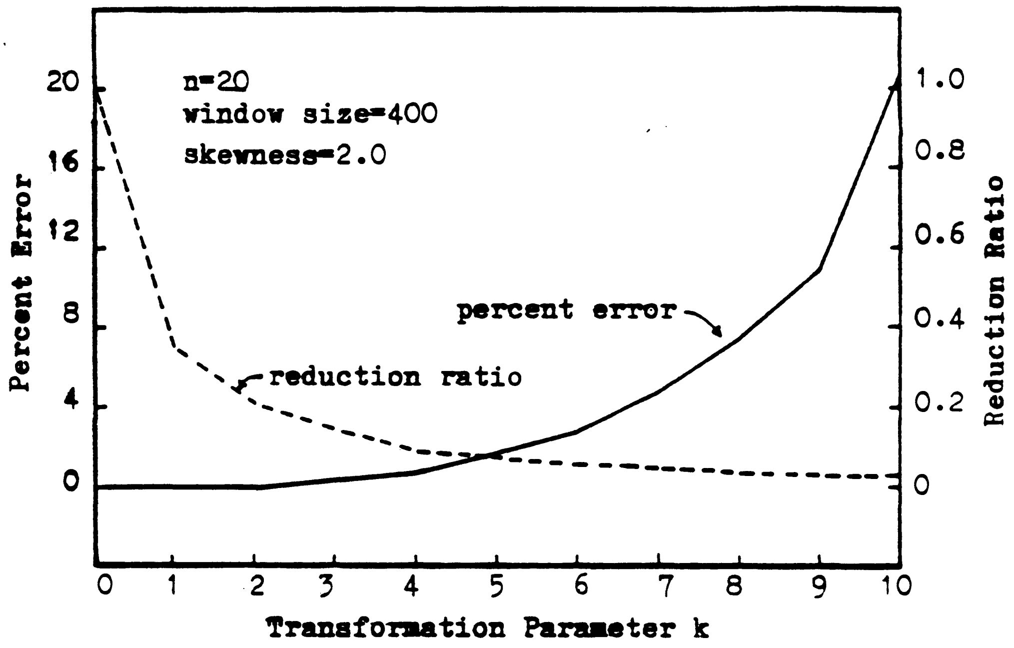

Fig. 3.2.2. Percentage error and reduction ratio

In an effort to identify the region of validity with respect to k, we plot the percentage error of $\bar{\omega}(\tau)$ and the reduction ratio (t_2^*/t_1) as a function of k in Figure 3.2.2. For the particular value of τ being used, the error in $\bar{\omega}(\tau)$ is very small for all $k \le 7$. These values of k

† According to Zipf's Law with skewness 2.0, the stack distance pmf is $\delta_i = c/i^{2.0}$, where c is a normalizing constant.

are such that the following inequality is satisfied

$$\tau' = (1-\Delta_k)\tau \geq n \qquad (3.2.4)$$

where n is the number of pages in the program. This condition parallels the one requiring memory sizes greater than k to be used in the study of fault rate statistics under LRU management. Fig. 3.2.2 also points out that a substantial reduction ratio in string length is obtained for this set of transformation parameter values.

The above results for the mean working set size are directly applicable to the steady-state fault rate observed under working-set memory management since one can express the fault rate as the first difference† of the mean working set size [Spi77a] :

$$\lambda(\tau) = \bar{\omega}(\tau) - \bar{\omega}(\tau-1).$$

4. Comparison with the Stack Deletion Method

In the previous section we have seen that the Stack Deletion Method and the proposed transformation method are very similar to each other. We note, however, that the page names associated with the references in the strings resulting from the compression of the GEN1 output by the Stack Deletion Method and the output of GEN2 are <u>not</u> the same. This can be explained by observing that while the transformation method generates no references to stack depths less than or equal to k, the Stack Deletion Method processes <u>all</u> of the references in the original string (in

† This is the discrete time analog of the first derivative.

this case, that produced by GEN1), thus causing stack updates at each reference. In otner words, the two methods produce results that have similar <u>distance</u> strings (the d_i's) but different <u>reference</u> strings (the r_i's).

TABLE I ǂ

RATIO OF GEN1 STRING LENGTH TO GEN2 STRING LENGTH (t_1/t_2^*)

Parameter k	WATFIV	FFT	APL
2	4.54	5.08	5.26
3	7.04	11.64	9.55
4	9.80	24.80	13.15
5	13.00	33.75	17.85
8	25.38	73.34	43.48
12	38.17	178.22	88.50

The reduction in string length due to both methods is given by $1/(1-\Delta_k)$. Table I shows the length reductions obtained when we apply the transformation method with various values of the parameter k to the LRUSM of three sample programs. More data about the traces from which the LRUSM for the three programs were obtained are presented by Smith [Smi76b, Smi77a].

ǂ Generated from data presented by Lau [Lau79a].

5. Applications

The application of the transformation method to generator-driven simulation studies within the framework discussed in section 2.3 is immediate. The simulators being driven by GEN2 process the references just as before, but increment the clock by the quantity R^* instead of by 1 at each reference. Choosing a value of k to use as the transformation parameter involves a tradeoff between the simulation speed-up desired and the range of validity of the results. Working-set management studies should be restricted to the set of window size values that satisfy inequality (3.2.4), while LRU studies are applicable only for memory sizes greater than k.

Although we have emphasized generator-driven simulation studies as the main application area, the method can also be used in synthetic program design for virtual-memory environments [Bab81a]. Recall that a synthetic program [Buc69a] is a parameterized piece of code that consumes controlled amounts of system resources. In a virtual-memory environment, programs consume resources (CPU cycles, disk I/O bandwidth, etc.) not only explicitly, but also implicitly (CPU cycles, main memory, and paging I/O bandwidth) because of the automatic memory management functions. One of the simplest ways to characterize the behaviour of a program in a virtual memory environment is to specify its <u>lifetime</u> function [Bel69a]. This function gives the mean number of instructions executed by the program between consecutive page faults when it is allocated a mean number of m pages. Suppose we extend the classical synthetic program requirements by specifying a lifetime function it has to conform to. We proceed by determining the parameters of the LRUSM corresponding to the given lifetime function †. The synthetic program is simply an implementation of the reference string generation algorithm that is based on this LRUSM. However, the execution of this synthetic program (observed at the memory reference level) results in a string like $q_1, q_2, \ldots, q_Q, r_1, q_1, q_2, \ldots, q_Q, r_2, \ldots$ The Q references (all to a small set of pages containing the code and data for the synthetic program) between each of the desired references (the r_i's) are due to random number generation, stack updating and other functions that the program has to perform. To be able to run the synthetic program in real time, these Q references are clearly undesirable and will be termed *overhead references*. Suppose that we apply our transformation method with parameter k to the LRUSM built into the program, with k being the smallest integer for which $R^* \geq Q$. Now, the Q overhead references appear to be part of the desired reference string and the synthetic program reproduces the paging behaviour specified by the input lifetime function within the applicability of the transformation method with parameter k in *real time*. Further details of the topic along with an actual implementation are discussed by Babaoglu [Bab81a].

6. Conclusions

We have presented a method for the efficient generation of memory reference strings based on the LRU stack model of program behaviour. The claim that the shorter output of the modified generator preserves the page fault rate characteristics of the original string (i.e., that

† Under LRU management, there exists a one-to-one correspondence between an LRUSM and a lifetime function that is nondecreasing.

produced by the unmodified generator) when processed by the LRU policy was shown analytically for memory sizes greater than k. The range of applicability under the working set memory-management policy is not as sharply defined. However, we have presented some necessary conditions that need to be satisfied for the fault rate and mean memory occupancy results to be valid.

The method provides a generator that is extremely economical both in space and in time and, whenever the LRUSM has been judged an adequate model of the characteristics that are of interest to the particular study, can be used as a source of memory references for any simulator that relies on trace data as input. Another interesting application of the method to the construction of synthetic programs for virtual-memory environments has also been briefly discussed.

Acknowledgments: The author wishes to thank Domenico Ferrari, Alan Smith and the members of the PROGRES (PROGram REStructuring) group at Berkeley for their valuable comments during the preparation of this paper.

The Applied Mathematical Sciences Research Program of the Office of Energy Research, Department of Energy is gratefully acknowleged for providing partial support for this work under contract W-7405-ENG-48.

7. References

[Aho71a] A. V. Aho, P. J. Denning, and J. D. Ullman, "Principles of Optimal Page Replacement," J. ACM Vol. **18** pp. 80-93 (January

1971).

[Ala80a] T. O. Alanko, I. J. Haikala, and P. H. Kutvonen, "Methodology and Empirical Results of Program Behaviour Measurements," Performance Eval. Rev. Vol. **9**(2) pp. 55-66 (Summer 1980). Proceedings of Performance 80

[Bab81a] Ö. Babaoğlu, "On Constructing Synthetic Programs for Virtual Memory Environments," pp. 195-204 in Experimental Computer Performance Evaluation, ed. Ferrari D. and Spadoni M., North-Holland, Amsterdam (1981).

[Bas76a] F. Baskett and A. Rafii, "The A_0 Inversion Model of Program Paging Behavior," Stanford U. Comp. Sci. Dept., Report STAN-CS-76-579 (October 1976).

[Bel66a] L. A. Belady, "A Study of Replacement Algorithms for a Virtual Storage Computer," IBM Syst. J. Vol. **5** pp. 78-101 (1966).

[Bel69a] L. A. Belady and C. J. Kuehner, "Dynamic Space Sharing in Computer Systems," Comm. ACM Vol. **12**(5) pp. 282-288 (May 1969).

[Buc69a] W. Buchholz, "A Synthetic Job for Measuring System Performance," IBM Syst. J. Vol. **8** pp. 309-318 (1969).

[Cof71a] E. G. Coffman and B. Randell, "Performance Prediction for Extended Paged Memories," Acta Informatica Vol. **1**(1) pp. 1-13 (1971).

[Cof73a] E. G. Coffman and P.J. Denning, Operating Systems Theory, Prentice-Hall, Enlewood Cliff, New Jersey (1973).

[Den68a] P. J. Denning, "The Working Set Model of Program Behavior," Comm. ACM Vol. **11**(5) pp. 323-333 (May 1968).

[Den80a] Peter J. Denning, "Working Sets Past and Present," IEEE Trans. Software Eng. Vol. **SE-6**(1) pp. 64-84 (January 1980).

[Fra78a] Mark A. Franklin, G. Scott Graham, and R. K. Gupta, "Anomalies With Variable Partition Paging Algorithms," Comm. ACM Vol. **21**(3) pp. 232-236 (March 1978).

[Kah76a] Kevin Kahn, "Program Behavior and Load Dependent System Performance," Ph.D. Th., Purdue U. Computer Science Department (May 1976).

[Knu73a] D. E. Knuth, The Art of Computer Programming - Sorting and Searching, Addison-Wesley, Mass. (1973). vol. 3

[Lau79a] E. Lau, "Performance Improvement of Virtual Memory Systems by Restructuring and Prefetching," Ph.D. Th., Univ. California Berkeley, California (1979).

[Lew71a] P. A. W. Lewis and P. C. Yue, "Statistical Analysis of Program Reference Patterns in a Paging Environment," Proc. IEEE Int. Comptr. Soc. Conf., pp. 133-134 (September 1971).

[Lew73a] P. A. W. Lewis and G. S. Shedler, "Empirically Derived Micro Models for Sequences of Page Exceptions," IBM J. Res. Develop. Vol. **17**(2) pp. 86-100 (March 1973).

[Mat70a] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," IBM Syst. J.

Vol. **9** pp. 78-117 (1970).

[Olk80a] F. Olken, "Efficient Methods for Calculating the Success Function of Fixed Space Replacement Policies," M.Sc. Report, Univ. of California Berkeley, California (1980).

[Pri74a] B. G. Prieve, "A Page Partition Replacement Algorithm," Ph.D. Th., Univ. of California Berkeley, California (1974).

[Raf76a] A. Rafii, "Empirical and Analytical Studies of Program Reference Behavior," Ph.D. Th., SLAC Report 197, Stanford California (1976).

[Rau77a] B. R. Rau, "Properties and Applications of the Least-Recently-Used Stack Model," Stanford U. Digital Syst. Lab. Report No. 139 (May 1977).

[Rau79a] B. R. Rau, "Program Behavior and the Performance of Interleaved Memories," _IEEE Trans. Comptrs._ Vol. **C-28** pp. 191-199 (March 1979).

[Ros70a] S. M. Ross, _Applied Probability Models with Optimization Applications_, Holden-Day, San Francisco (1970).

[Smi76a] A. J. Smith, "Analysis of the Optimal Look Ahead Demand Paging Algorithms," _SIAM J. Comptng._ Vol. **5** pp. 743-757 (December 1976).

[Smi76b] A. J. Smith, "A Modified Working Set Paging Algorithm," _IEEE Trans. Comptrs._ Vol. **C-25** pp. 907-914 (September 1976).

[Smi77a] A. J. Smith, "Two Simple Methods for the Efficient Analysis of Memory Address Trace Data," _IEEE Trans. Software Eng._ Vol. **SE-3** pp. 94-101 (January 1977).

[Smi81a] A. J. Smith, "Analysis of Long Term File Reference Patterns for Application to File Migration Algorithms," _IEEE Trans. Software Eng._ Vol. **SE-7**(4) pp. 403-417 (July 1981).

[Spi77a] Jeffrey Spirn, _Program Behavior: Models and Measurements_, Elsevier North-Holland, New York (1977).

[Spi72a] J. Spirn and P. J. Denning, "Experiments with Program Locality," _Proc. Fall Joint Comptr. Conf._, pp. 611-622 (1972).

[Spi76a] J. Spirn, "Distance String Models for Program Behavior," _Computer_ Vol. **9** pp. 14-20 (November 1976).

[Wal77a] A. J. Walker, "An Efficient Method for Generating Discrete Random Variables With General Distributions," _ACM Trans. Math. Software_ Vol. **3**(3) pp. 253-256 (September 1977).