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

by

Özalp Babaoğlu

Memorandum No. UCB/ERL M79/75

21 November 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

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

Özalp Babaoğlu

Computer Science Division
Department of Electrical Engineering and Computer Science
and the Electronics Research Laboratory
University of California, Berkeley
Berkeley, California 94720

ABSTRACT

We consider the problem of generating memory reference strings that are representative of real programs where the generator is based on the Least Recently Used (LRU) Stack Model of program behavior. A method to transform the stack distance probability mass function that drives the generator is proposed which results in memory reference strings that are considerably shorter in length while preserving most of the essential characteristics of the original string. 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 or equal to k, the parameter of the transformation.

Keywords and Phrases : paging, virtual memory, program behavior, LRU stack model, trace driven simulation, synthetic job.

November 21, 1979

† This material is based upon work supported 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 Behavior†

Özalp Babaoğlu

Computer Science Division
Department of Electrical Engineering and Computer Science
and the Electronics Research Laboratory
University of California, Berkeley
Berkeley, California 94720

1. Introduction

Trace driven simulation is a frequently used method for studying the performance of various aspects of paged virtual memory computer systems [BELL66, LAUE79, MATS70]. The trace data used in these studies consists of a record of all the memory addresses (data and instruction) generated during the execution of a program. The major drawback of this approach is that simulation studies dealing with realistically long trace data (a few million references) are very costly both in space and time. Smith [SMIA77] has studied two methods for reducing these costs by compressing the trace data while preserving its essential characteristics. 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 $k-1$ *most recently used* pages. The second method studied by Smith, the *Snapshot* or *Reference Set* [LAUE79, PRIB74] with sample interval T, on the other hand 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 characteristics (such as page fault rate and mean working set size) of the original reference string has been verified experimentally when they are processed by a wide variety of paging algorithms [SMIA77]. An alternative approach to reducing the space cost of such studies is to use a model of the program behavior such as the *Independent Reference Model* (IRM) [AHOA71, BASF76, SPRJ72] or the *Least Recently Used Stack Model* (LRUSM) [LENJ, RAUB77, SPRJ76] in conjunction with a random number source, thus obtaining a generative model [SPRJ77]. 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, arbitrary length memory reference strings can be generated one reference at a time with essentially no space cost. Note that from the viewpoint of the simulator, the trace driven and the generator driven methods are identical. Although extremely compact in space requirements, generator driven simulations are usually more costly in time than trace driven simulations. This is due to the fact that it usually requires more computation to *generate* a reference than to simply *read* it from the trace data.

In this paper we propose a method for improving the time cost of generator driven simulations by reducing the number of references generated. We show that the method is the parallel of the Stack Deletion Method for trace data compression in the realm of trace driven simulation.

† This material is based upon work supported by the National Science Foundation under Grants No. MCS 7824618 and MCS 7807291

2. The LRU Stack Model

The model of program behavior that our study will be based upon is the LRUSM . In this model, the n pages of the program are envisioned to be ordered in a stack according to their recency of usage. The page referenced at time t, r_t, is the one currently occupying stack position d_t, where the d_t's are independent and identically distributed random variables with $\Pr\{d_t=j\}=\alpha_j$ for all t. The set $\mathbf{D}=(\alpha_1, \alpha_2, \cdots, \alpha_n)$ where $\sum_{i=1}^{n}\alpha_i=1$ and $\alpha_i>0$ for all i uniquely defines the LRUSM. The vector **D** is often referred to as the *stack distance probability mass function* (pmf). Each 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 shift down one position. The set of pages occupying stack positions below d_t remain unaffected. Based on this description of the LRUSM, the Stack Deletion Method of trace data compression with parameter k can be characterized as removing from the trace all references that are to stack positions less than the parameter. Properties of the LRUSM based on analytic and experimental studies have been extensively reported [RAUB77, SMIA76b, SPRJ72]. These studies have indicated that the LRUSM is able to reproduce quite faithfully the page fault rate and the mean memory occupancy of programs under a variety of paging algorithms. More recently, Rau [RAUB79] has applied the LRUSM to modeling program module reference behavior in an interleaved memory environment with good results.

Constructing a generative model based on the LRUSM simply involves generating the d_i's as independent random samples from $(d_1, d_2, \cdots, d_n)$ according to **D** and performing the necessary stack manipulations to obtain the r_i's. One possible approach to reducing the time cost of a generator driven simulator is the following. The generator is allowed to run unaltered but references generated to pages at stack depths less than k are suppressed from the output. This scheme exactly corresponds to filtering the output of the generator through an implementation of the Stack Deletion Method with parameter k. Note however that *all* references in the original string are in fact generated although some are never seen by the simulator. What we seek is a method for directly generating a string that has the properties of the original string but the length of the compressed string.

3. The Transformation Method

3.1. Fault Rate Characteristics

Let $\mathbf{D}_1 = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be the driving stack distance pmf for the original generator (hereafter referred to as G1). For $\mathbf{D}_1$, we define the cumulative probabilities as $\beta_j = \sum_{i=1}^{j}\alpha_i$.

We know that the events corresponding to page faults for the page reference string generated by G1 in a memory of size m managed by the LRU policy constitute a discrete time renewal process where the inter-fault time distribution is geometric with parameter $\lambda = (1-\beta_m) = \sum_{j=m+1}^{n}\alpha_j$ [SPRJ76].

Let $N_1(m,t_1)$ be the number of faults generated by time t_1 for a memory of size m. By renewal theory [ROSS70], the steady state fault rate is given by

$$\lim_{t_1\to\infty}\frac{N_1(m,t_1)}{t_1} = (1-\beta_m) = \lambda \qquad (3.1.1a)$$

Furthermore, by the Elementary Renewal Theorem,

$$\lim_{t_1 \to \infty} \mathbf{E}\left[\frac{N_1(m,t_1)}{t_1}\right] = \lambda \tag{3.1.1b}$$

as well. Both equations (3.1.1a) and (3.1.1b) hold with probability 1.
For $t_1 < \infty$, however,

$$\mathbf{Pr}\left\{\left|\frac{N_1(m,t_1)}{t_1} - \lambda\right| \geqslant \epsilon\right\} \leqslant \frac{\sigma^2}{\epsilon^2} \tag{3.1.2}$$

where

$$\sigma^2 = Var(N_1(m,t_1)/t_1) = (\beta_m(1-\beta_m))/t_1 \tag{3.1.3}$$

for any $\epsilon > 0$.

Note that $\left[\frac{N_1(m,t_1)}{t_1} - \epsilon\ ,\ \frac{N_1(m,t_1)}{t_1} + \epsilon\right]$ is a level $(1 - \frac{\alpha^2}{\epsilon^2})$ confidence interval for λ.

Now consider a second generator (G2) that is driven by the following transformed stack distance pmf:

$$\mathbf{D}_2 = (\alpha'_1, \alpha'_2, \ldots, \alpha'_{k-1},\ R\alpha_k,\ R\alpha_{k+1}\ ,\ \cdots\ ,\ R\alpha_n)$$

where k is the parameter of the transformation and R is a constant.

Note that the transformation of α_i , $(1 \leqslant i \leqslant k-1)$ is left unspecified; the only constraint it has to satisfy is.

$$\sum_{j=1}^{k-1} \alpha'_j = 1 - R(1-\beta_{k-1}) \tag{3.1.4}$$

so that $\mathbf{D}_2$ is indeed a pmf.
As before, let $N_2(m,t_2)$ be the number of faults generated by time t_2 for a memory size m.

We are now in a position to state the transformation method.

Proposition : For all memory sizes $m \geqslant k$, the transformation with $R = R^* = 1/(1-\beta_{k-1})$ minimizes t_2/t_1 while preserving the size of the confidence interval of the steady state fault rate λ for any given confidence level.

Proof: Note that for G2, the statistic $\frac{N_2(m,t_2)}{Rt_2}$ is an unbiased estimate of λ.

For the transformation to preserve the length and level of the confidence interval for λ , it is necessary that

$$\Pr\left\{\left|\frac{N_1(m,t_1)}{t_1}-\lambda\right|\geqslant\epsilon\right\}=\Pr\left\{\left|\frac{N_2(m,t_2)}{t_2}-\lambda\right|\geqslant\epsilon\right\} \tag{3.1.5}$$

Substituting equations (3.1.2) and (3.1.3) into equation (3.1.5) and simplifying, we obtain

$$\frac{t_2}{t_1}=\frac{1-R(1-\beta_m)}{R\beta_m} \tag{3.1.6}$$

where we have also made use of the fact that $\alpha'_i = R\alpha_i$ since we only consider memory sizes greater than or equal to k.

Let $g(m,R) = t_2/t_1$ denote the running time ratio of G2 to G1. To minimize $g(m,R)$ as a function of R , we formulate the following optimization problem :

$$\begin{gathered}\text{minimize } g(m,R)\\ \text{subject to } 0\leqslant R\leqslant\frac{1}{1-\beta_{k-1}}\\ m\geqslant k\end{gathered}$$

where the constraints simply ensure that $\mathbf{D}_2$ is a valid pmf. The problem can be solved easily and has the solution $R = R^* = \dfrac{1}{1-\beta_{k-1}}$ □

The function $g(m,R)$ represents the ratio of the G2 string length to the G1 string length. From the above result, the minimum value for the function is given by

$$g(m,R^*)=1-\frac{\beta_{k-1}}{\beta_m}\leqslant 1,\quad m\geqslant k.$$

Note that

$$g(m,R^*)\leqslant\frac{1}{R^*}\leqslant 1$$

with the first equality holding when $m=n$. This observation leads us to the conclusion that the length of the string generated by G2 need only be at most $1/R^*$th of the length of that due to G1 to achieve the same size confidence interval of λ for a given confidence level and for all memory sizes greater than or equal to k.

Recalling the form of $\mathbf{D}_2$ and equation (3.1.4), the stack distance pmf transformation implied by the optimal value of R is of the form

$$\mathbf{D}_2=(0,0,\ldots,0,R^*\alpha_k,R^*\alpha_{k+1},\ldots,R^*\alpha_n)$$

We make the following observations about the transformation method:

[1] The transformation preserves the long run relative occurrances of stack depths greater than or equal to k i.e., $\alpha_i/\alpha_j = \alpha'_i/\alpha'_j$ for all i and $j \geqslant k$.

[2] G2 produces no references to stack depths less than k, while it references depths greater than $k-1$ 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^*, has the interpretation of the expected number of references until the first reference that is to a stack depth greater than $k-1$. This confirms our earlier observation that G2 suppresses those 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 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 is another important property associated with a reference string. As an example of a variable partition policy, we will consider the behavior of the output of G2 when processed by the working set [DENP68] algorithm.

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

$$\mathbf{Pr}\{\omega(\tau)=i\} = \beta_i \mathbf{Pr}\{\omega(\tau-1)=i\} + (1-\beta_{i-1})\mathbf{Pr}\{\omega(\tau-1)=i-1\} \tag{3.2.1}$$

where $\mathbf{Pr}\{\omega(1)=1\} = 1$

As applied to the LRUSM that drives G2, equation (3.2.1) becomes

$$\mathbf{Pr}\{\omega(\tau')=i\} = \beta'_i \mathbf{Pr}\{\omega(\tau'-1)=i\} + (1-\beta'_{i-1})\mathbf{Pr}\{\omega(\tau'-1)=i-1\} \tag{3.2.2}$$

where $\beta'_i = (\beta_i-\beta_{k-1})/(1-\beta_{k-1})$ and $\tau' = (1-\beta_{k-1})\tau = \tau/R^*$ since each reference generated by G2 advances the clock by R^* ticks rather than one. If α_i are non zero for all $i \geqslant k$, equation (3.2.2) which is valid only for $i \geqslant k$ has the closed form solution [LENJ]

$$\mathbf{Pr}\{\omega(\tau')=i\} = \sum_{j=k}^{i} \frac{(1-\beta'_k)(1-\beta'_{k+1})\ldots(1-\beta'_{i-1})}{\prod_{\substack{l=k \\ l\neq j}}^{i}(\beta'_j-\beta'_l)} {\beta'_j}^{\tau'-1}, \quad k \leqslant i \leqslant n \tag{3.2.3}$$

$$= 0, \quad i<k$$

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

In Fig. 3.2.1, we present the distribution of working set size that results for a sample LRUSM where the model parameters were generated through

$$\beta_i = (1-Ai^{-K})/(1-An^{-K})$$

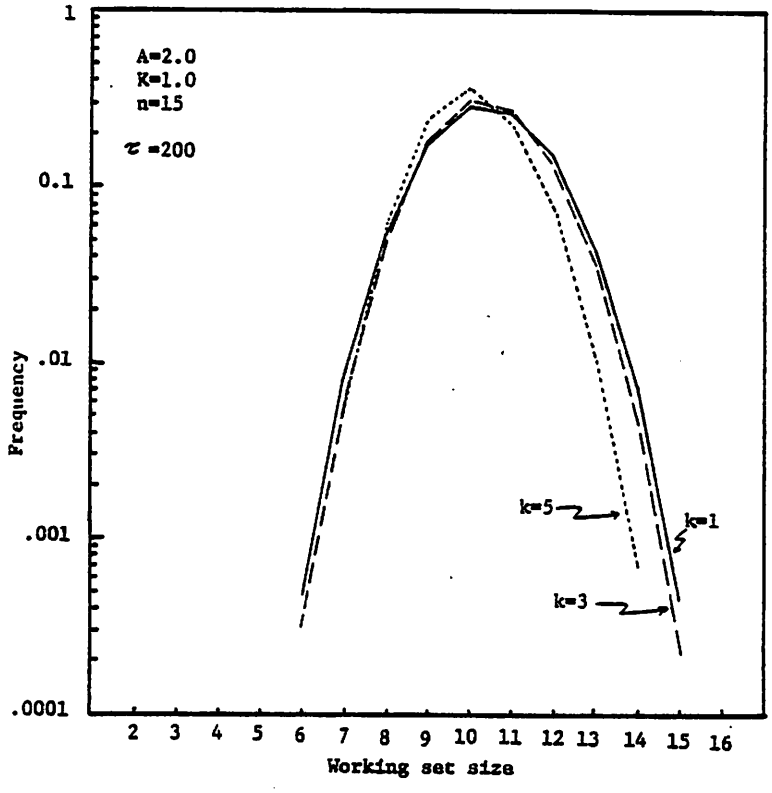

Figure 3.2.1 Working set size distribution of example LRUSM

where A and K are often known as the *lifetime* parameters of a program [SPRJ77].

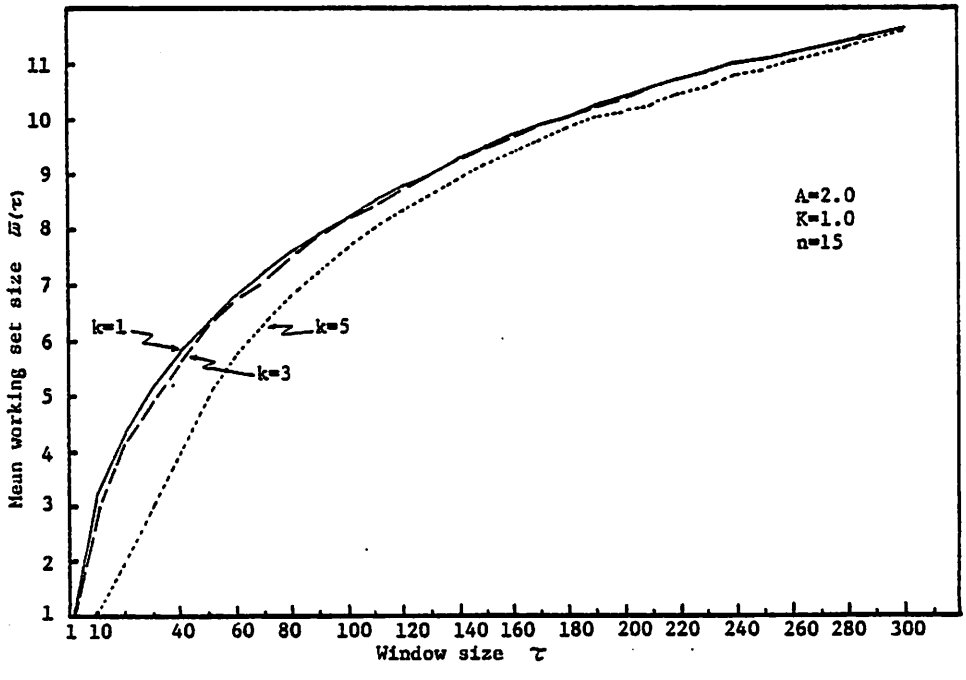

Figure 3.2.2 Mean working set size

Note that setting k=1 results in the null transformation and reduces G2 to G1. Fig. 3.2.1

indicates that the distortion of the working set size distribution increases proportional to k. In Fig. 3.2.2 however, we see that the transformation preserves the first moment of the distribution even for large k particularly when the working set parameter τ is large. 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 $\tau'=\tau/10.6$ when $k=5$).

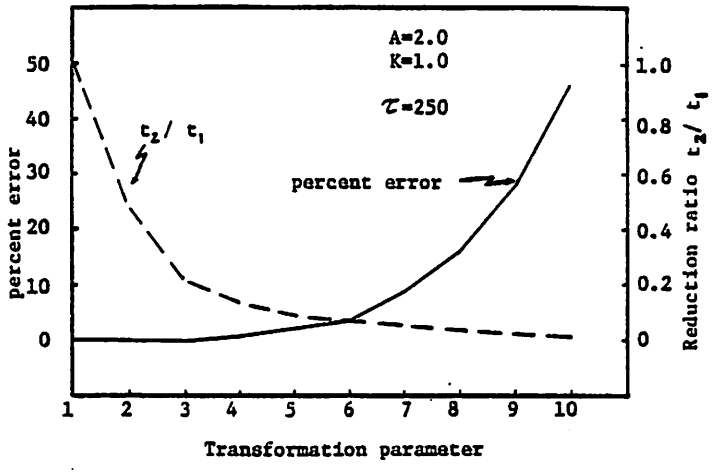

Figure 3.2.3 Percentage error and reduction ratio

In an effort to identify the region of validity with respect to k, we plot percentage error of $\bar{\omega}(\tau)$ and the reduction ratio $(g(n,R^*))$ as a function of k in Fig. 3.2.3. For the particular value of τ being used, the error in $\bar{\omega}(\tau)$ is very close to zero for all $k \leqslant 6$. These values of k are such that the following inequality is satisfied

$$\tau' = (1-\beta_{k-1})\tau \geqslant n \tag{3.2.4}$$

where n is the number of pages in the program. This condition parallels the one requiring memory sizes greater than $k-1$ to be used in the study of fault rate statistics under LRU management. Fig. 3.2.3 also points out that a substantial reduction ratio in string length is obtained for these 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 [SPRJ77]

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

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 two methods are *not* the same. This can be explained by observing that while the transformation method generates no

† This is the discrete time analog of the first derivative

references to stack depths less than k, the Stack Deletion Method processes *all* of the references in the original string, thus causing stack updates at each reference. In other words, the two methods produce results that have similar *distance* strings (the d_i's) but different *reference* strings (the r_i's).

TABLE I ‡
RATIO OF G1 STRING LENGTH TO G2 STRING LENGTH $(g(n,R^*)^{-1})$

Parameter k	WATFIV	FFT	APL
3	4.735	4.017	5.79
4	7.484	7.698	-
5	10.704	26.6	-
6	14.514	36.2	22.6
8	-	-	46.2
12	-	-	109.2

The reduction in string length due to both methods is given by $1/(1-\beta_{k-1})$. Table I indicates the length reduction 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 can be found in references [SMIA776a] and [SMIA77].

4.1. Simple Modifications

Through some simple refinements of the stack updating algorithm, the transformation method and the Stack Deletion Method can be made to have similar *reference* strings as well. As described, the reference string generated by the transformation method differs from that of the Stack Deletion Method simply because the top $k-1$ pages of the stack remain un updated. If, at the time of generating a reference, the true permutation of these $k-1$ pages were known, we could duplicate Stack Deletion exactly. In fact, since we are only interested in memory sizes greater than $k-1$, it suffices to know just the identity of the page occupying the $k-1$st position of the stack (this is one of the pages of the so called *push-pull pair* [COFE71]). Through the following analysis, we can derive a probability distribution for the identity of this page amongst those currently in the top $k-1$ positions of the stack.

Consider the Markov chain where the state space consists of all possible permutations of the integers $1,2,\ldots,k-1$ corresponding to the relative ordering of the pages currently occupying the top $k-1$ stack positions. Given state $\mathbf{s}_i=\{s_1,s_2,\ldots,s_{l-1},s_l,s_{l+1},\ldots,s_{k-1}\}$, the transition probability to state $\mathbf{s}_j$ is α_l if and only if $\mathbf{s}_j=\{s_l,s_1,s_2,\ldots,s_{l-1},s_{l+1},\ldots,s_{k-1}\}$. Let $\mathbf{P}$ denote the single-step transition probability matrix for this chain. Given an initial state $\mathbf{s}_i$, what we are interested in is the probability that the page currently at position l will be in position $k-1$ after exactly R references to the top $k-1$ slots of the stack. This is given by

$$\Pr\{s_{k-1}=l \text{ after } R \text{ references} \mid \mathbf{s}_i\} = \sum_{\mathbf{s}_j \in \Theta} \mathbf{P}^R_{\mathbf{s}_i,\mathbf{s}_j}$$

where Θ is the set of $(k-2)!$ states such that $s_{k-1}=l$ and $\mathbf{P}^R_{\mathbf{s}_i,\mathbf{s}_j}$ is the $(\mathbf{s}_i,\mathbf{s}_j)$th element of the R-step transition probability matrix.† Having obtained the distribution, the page that is to be pushed down to the k th stack position is selected amongst those currently in the top $k-1$ slots

‡ Reproduced from [SMIA77]

† The R-step transition probability matrix, $\mathbf{P}^R$, is obtained by raising the single-step transition probability matrix, $\mathbf{P}$, to the Rth power. Being strictly a function of the original LRUSM parameters, this matrix need be computed only once and stored for repeated use.

according to this distribution. Otherwise, the method is not altered.

Due to factorial growth, for large k, computing and storing $\mathbf{P}^R$ can be rather expensive. If R is large, we can approximate the R-step probabilities with the steady state probabilities and simply the problem considerably. Since $\mathbf{P}$ is *doubly stochastic*,‡ all steady state stack permutations are equally likely, thus the page to be pushed can be selected uniformly over those in the top $k-1$ stack positions without having to go through the above computations.

Another modification of the transformation method is suggested by the observation that it results in mean working set sizes that are consistently smaller than those for LRUSM. This is a consequence of not referencing pages in the top $k-1$ stack positions. This error becomes substantial as the reduction ratio, R^*, approaches the window size, τ. Use of the Snapshot Method, whereby any page in the top $k-1$ positions of the stack which has not appeared in the trace in the last T time units is issued as a memory reference, along with the transformation method can reduce this error even for small τ.

5. Applications

The application of the transformation method to trace driven simulation studies is immediate. The simulators being driven by G2 process the references just as before except for incrementing the clock by the quantity R^* as opposed to 1 per reference. Chosing a value of k to use as the transformation parameter involves a tradeoff between 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 $k-1$.

Although we have emphasized trace driven simulation studies as the main application area for the method, it is potentially suitable for the area of synthetic job [BUCW69] design for virtual memory environments. Such a job is a highly parameterized program that consumes controlled amounts of system resources. In a virtual memory environment, one consumes resources (CPU cycles, disk I/O bandwidth, etc.) not only explicitly, but also implicitly (main memory and paging I/O bandwidth) through the pattern of memory addresses generated. To vary the memory reference behavior of a program in a controlled manner it is required that the program have a model for the action built into it. Consider the page reference string $r_1, r_2, r_3, \ldots$ where $r_i \in \{1, 2, \ldots, n\}$, whose characteristics (as we have defined them) we would like the synthetic job to replacate. The execution of the synthetic job (observed at the memory reference level) that has simply an LRU stack model of the above string built into it results in $q_1, q_2, \ldots, q_Q, r_1, q_1, \ldots, q_Q, r_2, \ldots$ The Q references (all to a small set of l pages containing the code and data for the synthetic job) between each of the target references are due to the random number generation, distribution transformation, stack updating and other functions that the program has to perform. To be able to run the synthetic job in real-time, these Q references are clearly undesirable and will be termed *interference references.* Suppose that we apply the transformation method with parameter k to the LRUSM built into the program such that k is the smallest integer for which $R^* \geq Q$ and $k \geq l$. Now, the Q interference references appear to be part of the target reference string and the synthetic job reproduces the paging behavior specified by the original LRUSM within the applicability of the transformation method with parameter k in *real-time.*

‡ A probability matrix $\mathbf{P}$ is doubly stochastic if $\sum_j P_{i,j} = 1$ and $\sum_i P_{i,j} = 1$ for all i and j.

6. Conclusions

We have presented a method for the efficient generation of memory reference strings based on the LRU stack model of program behavior. The claim that the output of the modified generator preserves the page fault rate characteristics of the original string when processed by the LRU policy was proven for memory sizes greater than or equal to k. The range of applicability under the working set memory management policy is not as sharply defined. However, we have presented 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 can be used as a source of memory references for most simulators that rely on trace data as input. Another interesting application of the method to the construction of synthetic jobs for virtual memory environments has also been briefly discussed.

Acknowledgements. I am grateful to Domenico Ferrari, Makoto Kobayashi and the other members of the PROGRES group for their valuable suggestions during this study. I would also like to thank Alan Smith and Juan Porcar for commenting on an earlier version of this paper.

References

[AHOA71] A. V. Aho, P. J. Denning and J. D. Ullman, "Principles of Optimal Page Replacement," *J. Ass. Comput. Mach.* , vol. 18, pp. 80-93, Jan. 1971.

[BASF76] F. Baskett and A. Rafii, "The A_0 Inversion Model of Program Paging Behavior," Stanford Univ. Comp. Sci. Dept. Report STAN-CS-76-579, Oct. 1976.

[BELL66] L. A. Belady, "A Study of Replacement Algorithms for a Virtual Storage Computer," *IBM Syst. J.* , vol. 5, pp. 78-101, 1966.

[BUCW69] W. Buchholz, "A Synthetic Job for Measuring System Performance," *IBM Syst. J.* , vol. 8, pp. 309-318, 1969.

[COFE73] E. G. Coffman and B. Randell, "Performance Prediction for Extended Paged Memories," *Acta Informatica,* 1, 1, pp. 1-13, 1973.

[DENP68] P. J. Denning, "The Working Set Model of Program Behavior," *Commun. Ass. Comput. Machinery,* vol. 11, no. 5, pp. 323-333, May 1968.

[LAUE79] E. Lau, "Performance Improvement of Virtual Memory Systems by Restructuring and Prefetching," Ph.D. dissertation, Univ. California, Berkeley, CA. 1979.

[LENJ] J. Lenfant, "Comparison of the Working Sets and Bounded Locality Intervals of a Program," Inst. de Recherche en Informatique et Systeme Aleatoires, Research Report no. 41

[MATR70] 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.

[PRIB74] B. G. Prieve, "A Page Partition Replacement Algorithm," Ph.D. dissertation, Univ. California, Berkeley, CA. 1974.

[RAUB77] B. R. Rau, "Properties and Applications of the Least-Recently-Used Stack Model," Stanford Univ., Digital Syst. Lab. Report no. 139, May 1977.

[RAUB79] B. R. Rau, "Program Behavior and the Performance of Interleaved Memories," *IEEE Trans. Comput.*, vol. C-28, pp. 191-199, March 1979.

[ROSS70] S. M. Ross, *Applied Probability Models with Optimization Applications* , San Francisco: Holden-Day, 1970.

[SMIA76a] A. J. Smith, "A Modified Working Set Paging Algorithm," *IEEE Trans. Comput.* , vol. C-25, pp. 907-914, Sept. 1976.

[SMIA76b] A. J. Smith, "Analysis of the Optimal, Look Ahead, Demand Paging Algorithms," *SIAM J. Computing,* vol. 5, pp. 743-757, Dec. 1976.

[SMIA77] A. J. Smith, "Two Simple Methods for the Efficient Analysis of Memory Address Trace Data," *IEEE Trans. Soft. Eng.* , vol SE-3, pp. 94-101, Jan. 1977.

[SPRJ72] J. Spirn and P. J. Denning, "Experiments with Program Locality," *Proc. Fall Joint Comput. Conf.* pp. 611-622, 1972

[SPRJ76] J. Spirn, "Distance String Models for Program Behavior," *Computer,* vol. 9, pp. 14-20, Nov. 1976.

[SPRJ77] J. Spirn, *Program Behavior: Models and Measurements* , Elsevier North Holland, 1977.