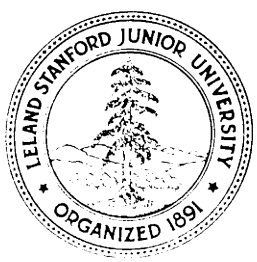
THE A∅ INVERSION MODEL OF PROGRAM PAGING BEHAVIOR

by

Forest Baskett and Abbas Rafii

STAN-CS-76-579
NOVEMBER 1976

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY



SLAC PUB-1826
STAN-CS-76-579

October 1976

THE AØ INVERSION MODEL OF PROGRAM PAGING BEHAVIOR

by

Forest Baskett
and
Abbas Rafii

Stanford Linear Accelerator Center*
P.O. Box 4349, Stanford, California 94305

and

Computer Science Department**
Stanford University, Stanford, California

Abstract

When the parameters of a simple stochastic model of the memory referencing behavior of computer programs are carefully selected, the model is able to mimic the paging behavior of a set of actual programs. The mimicry is successful using several different page replacement algorithms and a wide range of real memory sizes in a virtual memory environment. The model is based on the independent reference model with a new procedure for determining the page reference probabilities, the parameters of the model. We call the result the AØ inversion independent reference model. Since the fault rate (or miss ratio) is one aspect of program behavior that the model is able to capture for many different memory sizes, the model should be especially useful for evaluating multilevel memory organizations based on newly emerging memory technologies.

*Work partially supported by the Energy Research and Development Administration under Contract E(043)515.

**Work partially supported by National Science Foundation, NSF Grant GJ35720

KEYWORDS AND PHRASES

program models
stochastic program models
program behavior
program paging behavior
program page reference behavior
paging algorithms
replacement algorithms
virtual memory

THE AØ INVERSION MODEL OF PROGRAM PAGING BEHAVIOR

1. Introduction

Computing systems in which several types of storage are automatically made to appear as one uniform type of storage are likely to be a major part of our computing environment for some time to come. Memory transparency or automatic folding or virtual memory has been accepted as a necessary tool for the convenient solution of many computing problems in much the same way as higher level languages were accepted as a necessary evil many years ago. In fact, paging techniques [8] are being used to automatically manage small, very high speed buffers (caches) for high speed CPU's [2, 7] and to manage large, slow disk buffers for much larger and slower automated filing systems [3], as well as being used more conventionally to automatically manage main memory in a wide variety of computers.

This wide use of paging techniques, together with the ever changing performance parameters of the memory technologies on which these paging techniques are implemented, point up the need for efficient and effective methods for evaluating the performance of different memory hierarchy designs. Central to such methods will be some model of how computer programs reference memory. The choice of that model of memory referencing behavior will determine the accuracy, efficiency, generality, and even the feasibility of the evaluation method in which it is contained. In this paper, we describe a new interpretation of a simple model of how programs reference memory and give a procedure for determining the parameters of that model. We then illustrate the success of the model in predicting the page fault rate and working set

characteristics of actual programs by comparing the model predictions with results determined by simulations using actual program traces. We discuss some of the limitations of our method, how the results from our method compare with results from other models, and how the results may be useful. Finally, we discuss some possible extensions and generalizations.

2. Choice of model

The most widely used models of the memory referencing behavior of programs have been simulation models driven by traces of the addresses generated by actual programs. While these methods have the most potential for accuracy they are also the most cumbersome, expensive to process, and time consuming. In addition, it is normally not feasible to evaluate more than a small subset of the possible memory and system configurations of interest because of the difficulty in handling this type of model. Thus, one of our primary aims in developing an alternative model is to choose one which is analytically tractable. We want to be able to derive general results simply by solving equations involving the parameters of the system to be evaluated and the parameters of the model. Even if such solutions must be numerical instead of closed form expressions, we will have a model with more power than a simulation based on program traces. This will be power to investigate larger subsets of the memory design space more economically.

In addition to developing a model which is analytically tractable, we want a model with predictive power. We don't want to simply engage in curve fitting. We want the model to predict properties of how program references memory, which were not built into the model, through the method of determining the parameters. For example, the LRU stack depth model [10] of a program can predict the fault rate of the program under LRU page replacement precisely

if the stack depth distribution of the model is determined from the stack depth distribution of the program. Such a prediction is not surprising; it is built into the model. If, however, the model also predicted the fault rate with some accuracy under a different page replacement policy, we would say that the model had some predictive power.

Finally, we want a model which is sufficiently simple to be used efficiently in simulations since we expect to have to resort to simulations for the evaluation of some complex designs, or to validate simplifying assumptions in the analysis of some designs.

We concentrate on the paging behavior of the model of how programs reference memory since the principal performance characteristics of an automatic memory hierarchy can be determined from the paging behavior if we take a general view of paging. Thus when evaluating a cache memory design, the miss ratio is equivalent to the fault rate, the address mapping scheme corresponds to some special page replacement policy, and the size of cache elements is the page size. When evaluating an automated filing system, similar parallels can be drawn.

3. Notation

In order to develop and validate our model, we will refer to several different page replacement algorithms, the algorithms which specify which page in main memory is to be replaced when a program refers to a page in the backing store. We now give a brief summary of those algorithms and our notation for them.

MIN [5] -- replace the page whose next reference is furthest in the future. This algorithm is not practical because it requires knowledge of the future, but it is an algorithm which minimizes the total number of page

faults for a single program in a fixed size memory and, thus, is useful as a base for evaluating other page replacement algorithms. In addition, we make use of it in our procedure for determining the parameters of our model.

LRU [17] --replace the page which is Least Recently Used. This is the page whose last reference was furthest in the past. If the future references of a program are like the past references, then this time reversed dual of the MIN algorithm should be a good practical page replacement algorithm. In fact, it is difficult to keep track of which page was least recently used in real systems, but there are some simple and practical schemes which closely approximate the LRU page replacement policy [15]. Hence our model should be able to predict paging behavior under the LRU algorithm if it is to be useful in such settings.

FIFO [17] --replace the page which was first brought into main memory among those currently present in main memory (First-In-First Out). This algorithm is actually used in some computing systems [1] although it is known to have some strange properties and to generally be inferior to LRU and related algorithms. We consider it briefly as an extreme test.

AØ [4] --replace the page which is least likely to be referenced. This algorithm is useful when pages are known to be referenced with independent, fixed probabilities, as in the independent reference model. In such circumstances, it is known to be optimal among algorithms without knowledge of the future. We make extensive use of its analytic tractability in deriving the parameters of our model.

WS (T) [9] --replace the page which hasn't been referenced in the last T references. Those pages that have been referenced in the most recent T references are called the current working set. T (or tau) is the working set parameter. Of the page replacement policies we consider, this is the only one

which requires a variable number of main memory page frames. Thus, the paging behavior under the working set algorithm seems to represent a somewhat different dimension of program paging behavior than the previous fixed memory page replacement algorithms. Comparisons of our model results with actual working set results illustrate both the power and the limitations of our model.

4. Previous Models

Although simulation models based on actual program traces have been the most widely used models of memory referencing, numerous attempts have been made to develop more tractable analytic models. The two most prominent such models are the independent reference model and the LRU stack depth model.

In the independent reference model, there is a fixed probability, p_i, associated with each page i of the program being modeled. References to pages are generated by independently sampling from this page reference probability distribution. This model has received attention from a theoretical perspective because it is analytically tractable. There have been a number of papers giving interesting theoretical results based on this model [13,12]. The main problem with this model is the problem our intuition suggests, namely, that a program's references to memory are not independent but, in fact, are correlated in a complex and highly structured way. Thus, if we simply count the number of times a program references each of its pages, and use these counts as estimates of the page reference probabilities in an independent reference model of that program, we will find that the resulting model of the program is a very poor predictor of the actual paging behavior of the program. The fact that certain sets of pages tend to be referenced together (localities)

is not captured in this "page reference frequency" version of the independent reference model. Thus, the model predicts a page fault rate much higher than actually observed under almost all circumstances. This is illustrated in Figure 1 where we have plotted the actual page fault rate as a function of main memory size (in pages) for an IBM/360 WATFIV compiler run subject to the LRU and MIN page replacement algorithms. These functions are represented by the solid lines in Figure 1. On the same figure, we have plotted the fault rates observed for the page reference frequency independent reference model of this WATFIV compiler for the same memory sizes and page replacement policies. When we notice that the fault rate is plotted on a logarithmic scale, the overestimate of the model is most startling, usually between two and three orders of magnitude! Results like these, which are typical of the page reference frequency model, are why the independent reference model has often been harshly criticized by those interested in models that have some practical value.

The LRU stack depth model works from a probabilistic model of the depth of a reference in an LRU stack. The LRU stack is a stack in which the most recently referenced page is on the top of the stack, the next most recently referenced page is just below the top, down to the least recently used page on the bottom of the stack. Each time a page is referenced, the LRU stack is updated by moving the entry for that page from its current position (depth) in the stack to the top of the stack. This stack is for all of the pages referenced, not just those in main memory. We can maintain an LRU stack (in theory) even if we are not using an LRU page replacement policy. The LRU stack depth model is constructed by counting the number of times a particular position (depth) in the LRU stack is accessed in order to update the stack. These counts are then used as estimates of the probability of any given

reference being at a particular LRU stack position (depth). The LRU stack depth model can then be used to generate page references by generating a stack depth according to the stack depth distribution, looking in an LRU stack at that position, calling the page name found there the next page to be referenced, updating the LRU stack, and then repeating this process. The page reference string so generated does not have to be used in an LRU paging environment although the page fault rate for such a model will exactly match (except for sampling error) the LRU fault rate of the program from which it was derived.

A great deal has been written about efficient methods for determining the LRU stack depth distribution (and features of other stack processing type paging algorithms) [14,6] , but very little has appeared indicating the suitability (or lack of it) of the LRU stack depth model for systems other than LRU type paging environments. We will return to this subject later. The LRU stack depth model does not seem to be as analytically tractable as the independent reference model (the LRU stack depth model is an independent reference model in the strict sense of that term) although there are some available results. For example, the position of any particular page in the LRU stack of an LRU stack depth model is a uniformly distributed random variable independent of the identity of the page and the particular stack depth distribution [6]. Since this violates our intuition about program behavior, perhaps this partly explains why the LRU stack depth model has received little empirical treatment in the literature.

In addition to these two models, there have been fragmentary treatments of other types of models. We say that the treatments are fragmentary because they normally don't provide either enough theoretical development to demonstrate analytic tractability or enough empirical development to demonstrate practical

feasibility or value. They usually seem to be the starting points for more extensive research. One such model is a first order Markov model [16]. The independent reference model is sometimes called a zeroth order Markov model since successive references are independent of all past and future references. In a first order model, the next reference depends on the identity of the previous reference. A first order model should be able to do everything that a zeroth order model does plus more, provided the much larger parameter space can be algorithmically and efficiently determined. Models which explicity try to capture the idea of locality have been proposed [10], but not well developed.

5. AØ Inversion --Model

If we return to Figure 1 and the comparison of the fault rates for an actual program and the page reference frequency model of that program, we can see support for an observation that Peter Franazcek made to us [11]. He observed that while the page reference frequency model didn't predict actual fault rates very well, it did predict the relative performance of different paging algorithms with some accuracy. Thus, the spacing between the MIN and LRU curves is about the same for the actual program curves and the model program curves, indicating approximately the same percentage change in fault rate in the model as in the actual program between these two paging algorithms. This observation suggested to us that either the independent reference model was capturing some aspect of program behavior or that some paging results depended more on the system than on the program. At any rate, it seemed worthwhile to take another look at the independent reference model.

Our view of the model is a more abstract one than the view represented by the page reference frequency version of the model. We don't insist on any

particular identification between model pages and actual pages; we just insist on good results. Thus, we simply want to choose the page reference probabilities of the model so that the model more accurately predicts the actual fault rates in at least some cases. In this view an independent reference model of a program that references n pages is a model with n-1 parameters or degrees of freedom. With this many parameters, we should at least be able to do curve fitting provided we can devise a feasible scheme for determining the values of the parameters.

We observe that one way of binding the model to the characteristics of a real program is to require that the lower bound on the page fault rate of both the model and the actual program under optimal replacement algorithms be close together. We can then be sure that enough structure is built into the model so that, at least in the long run, the model is capable of predicting the behavior of the actual program under the optimal paging algorithm.

For a given page reference sequence of an actual program, we know that the MIN algorithm gives the least number of faults among all fixed memory size algorithms. We can, in fact, measure the MIN fault rate of the program, $F_{MIN}(m)$, for different memory sizes m ($1 \leq m \leq n$). For the independent reference model, the AØ [4] algorithm gives the optimal fault rate if we disallow the look-ahead of the MIN algorithm. At the time of a page fault, the AØ algorithm replaces a page which is least likely to be referenced in the future.

Let $[p_1, p_2, p_3, \ldots, p_n]$ and $p_1 \leq p_2 \leq p_3 \ldots \leq p_n$ be the set of reference probabilities for an independent reference model. The AØ fault rate produced by this model, $F_{A\emptyset}(m)$, for a memory size m is equal to [6] :

$$F_m = F_{A\emptyset}(m) = \sum_{i=m}^{n} P_i \quad - \quad \frac{\sum_{i=m}^{n} p_i^2}{\sum_{i=m}^{n} P_i} \qquad 1 \leq i \leq n \qquad (1)$$

Therefore, if the reference probabilities are known, (1) can give the optimal (non-lookahead) fault rate for different values of m, $1 \leq m \leq n$. Conversely, if a set of n fault rate values are given, we may be able to find a set of reference probabilities which satisfy the relations in (1).

We observe that if our independent reference model is to capture the fault rate behavior of actual programs, then we expect that the fault rate of the model under the AØ algorithm should be close to the fault rate of the actual program under the MIN algorithm and for all memory sizes. This gives us a procedure to find p_i's from the relations in (1). In other words, we now substitute for F_m's in (1) the observed MIN fault rate values, and then we invert (1) to get a set of recurrence expressions for finding p_i's. The independent reference model which is obtained by this procedure is referred to henceforth as the AØ inversion model.

We carry out this procedure by letting $S_m = \sum_{i=m}^{n} p_i$ and $R_m = \sum_{i=1}^{m} p_i$. We then successively get:

$$F_m = \sum_{i=m}^{n} p_i - \frac{\sum_{i=m}^{n} p_i^2}{\sum_{i=m}^{n} p_i}$$

$$F_m = S_m - \frac{\sum_{i=m}^{n} p_i^2}{S_m}$$

Similarly:

$$F_{m+1}S_{n+1} = S_{m+1}^2 - \sum_{i=m+1}^{n} p_i^2$$

Subtracting the above two expressions we get:

$$F_m S_m - F_{m+1}S_{m+1} = S_m^2 - S_{m+1}^2 - p_m^2$$

$$F_m p_m + F_m S_{m+1} - F_{m+1}S_{m+1} = p_m^2 + S_{m+1}^2 + 2p_m S_{m+1} - S_{m+1}^2 - p_m^2$$

or

$$p_m = \frac{S_{m+1}\,(F_m - F_{m+1})}{2S_{m+1} - F_m} \qquad 1 \le m \le n \qquad (2)$$

If p_n is known, then (2) can be used successivley to find p_{n-1}, p_{n-2} and so on. However, we can arrange (2) so that first we can find p_1, and having p_1, we can find p_2, and so on. Since p_i's are probabilities, we have

$$S_{m+1} = 1 - R_m = 1 - R_{m-1} - p_m .$$

Replacing this in (2), we find:

$$p_m = \frac{(1-R_{m-1}-p_m)\ (F_m-F_{m+1})}{2(1-R_{m-1}-p_m) - F_m} \qquad 1 \leq m < n \qquad (3)$$

In (3), we assume that $R_0 = 0$. Each p_i, i=1,2,3,...,n-1 can be successively computed from (3) by solving a quadratic equation. Later in this paper, we return to this derivation for more comments.

6. Test Results - Fault Rate Prediction

We now examine the ability of the AØ inversion model to predict the fault rate behavior of real programs. We expect to get substantial improvement over the previously mentioned page reference frequency method. Indeed, by inspecting Figure 2, we can see the success of the model. In this figure, the solid lines represent the fault rate curves of WATFIV program under MIN and LRU algorithms. Using the AØ inversion technique, we construct an independent reference model based on the same program. The MIN and LRU fault rate which are produced by the model are shown by dotted lines on the same figure. As we expected, the MIN fault rate curve of the model closely follows the MIN fault rate curve of the actual program for a wide range of memory sizes. It is interesting, however, that even the LRU fault rate curves of both the model and the acutal program are fairly close together. The success of the model becomes more significant if we compare Figure 1 with Figure 2 to see the amount of improvement over the page reference frequency method. This demonstrates the fact that by using an appropriate method, we can build substantial predictive power into a simple independent reference model.

It is interesting to inspect the set of reference probabilities which are obtained by the AØ inversion model. We can get a better insight into the structure of this model by comparing these reference probabilities with the reference

probabilities which are obtained from the simple frequency method. In Figure 3, the two sets of reference probability densities based on the WATFIV compiler are shown. The horizontal axis is the page number and the vertical axis is the probability weight.

In the frequency method, the reference probabilities are found by taking the global averages on the entire string. In the averaging process, most of the information about the regional characteristics of the string is lost. Along the same lines, we have tried other approaches to get a better representative set of probabilities. One method we used was to divide the trace into intervals and find the relative reference frequencies in each interval, and order each set and combine over all intervals. The results, which are not reported here, showed only a slight improvement over the usual frequency method.

In the AØ inversion model, a completely different approach is taken and the reference probabilities which are obtained in this case bear no direct relation with the relative reference frequency of each page in the actual program. In Figure 3, we note that the AØ inversion model produces a reference probability mass distribution which has a distinctive resemblance to the fault rate curve of the program upon which the model is based. We can see that some important information, such as the memory sizes where the actual fault rate changes curvature, is precisely carried over to the corresponding page numbers in the reference probability curve.

Generally, the AØ inversion model assigns large probability mass to a small number of pages (i.e., pages with the lowest subscript) and the remaining pages receive probability weights in sharply decreasing quantities. One can interpret the top pages (e.g., the first 20 pages in Figure 3) as the current locality pages of the program. References to these pages are mostly favored in the reference string generated by the model. The pages which receive the least probability weights can be imagined to produce the instances corresponding

to locality transitions in the actual program. The remaining pages which receive probability weights between the above two extremes can be considered to contribute to the small variation of the locality sizes in time.

We can support our claim about the predictive power of the AØ inversion model by presenting more evidences about the success of the model. For another replacement algorithm, we test the behavior of the model under the FIFO paging algorithm. In Figure 4, the solid line is the fault rate of the actual WATFIV program versus memory sizes under the FIFO algorithm. In the same figure, the dotted line represents the fault rate curve of the model under the same algorithm. We can see that the model is capable of predicing the average fault behavior of the program on the lower range of memory capacities. For very large memory sizes, the dotted line drifts slightly away from the solid line. The behavior of the model in this region can be partially accounted for by any one of the following reasons. Since we simulate the model, in this case the sampling error becomes significant for large memory sizes. The other source of the error is the inaccuracy in defining the tail (i.e., the pages with the highest subscripts) page reference probabilities. We shall return to the problem of finding the tail probabilities later in this paper.

In a series of experiments, we present more data for validation of the model. We have constructed AØ inversion models based on the page reference trace of several programs. These programs include a trace of a WATFIV compiler, a FORTRAN program called WATEX, an APL program, and the trace of a program to calculate the Fast Fourier Transform, called FFT, of a set of data points.

In Figures 5, 6, and 7, the fault rate curve of each model under the MIN and LRU algorithms are compared with those of the corresponding actual programs. In each figure, the solid lines belong to the actual program and the dotted lines represent the data points from the model. We note that in each case the

model is able to predict the LRU fault rate of the actual program in a satisfactory way. All these models are especially successful in the range of lower memory sizes. It is significant, for instance, to note that the AØ inversion model has been able to capture the special behavior of the FFT program as can be seen in Figure 7. We observe that the fault rate curves of the model breaks in exactly the right point (memory size), in this case. This is a rather promising result which shows that the technique can be used successfully to model program behaviors which are highly structured.

7. Average Working Set Size Prediction

The working set concept has been widely acclaimed as being a good measure of program reference localities. The working set [9], WS(t,T), at time t, is the set of pages addressed in the past T references. The size of this set is denoted by ws(t,T). The window size T is the working set parameter. The measured working set sizes can be averaged over the entire program trace and lumped into one number, called the average working set size, ws(T).

The average working set size can also be defined for the references generated by the model. Since the probabilistic structure of the model is known, the expected working set size can be readily obtained by a probalistic argument. Let $[p_1, p_2, p_3, \ldots, p_n]$ be the parameters of the AØ inversion independent reference model. The expected working set size with parameter T is equal to the probability that a page is in the working set summed over all pages. A page is in the working set if it has been referenced at least once in the last T units of time; therefore,

$$ws(T) = \sum_{i=1}^{n} \left[1 - (1-p_i)^T\right] \qquad (4)$$

We can now examine the capability of the model in predicting the average working set sizes of actual programs. In a series of experiments, we have mea-

sured the average working set sizes of a number of programs with different window sizes. For each actual program, the average working set sizes of the corresponding AØ inversion model is calculated from (4). The results are illustrated in Figures 8 and 9 for the WATFIV, APL and FFT programs. In each figure, the horizontal axis is the window size in terms of address reference units and the vertical axis is the average working set size. The solid lines are obtained from the measurements on the actual programs and the dotted lines are computed from the parameters of each model.

We can see that the predicted average working set size values derived from the model are strikingly close to those of the actual programs. This result demonstrates the capability of the AØ inversion model in capturing an important feature of the address reference behavior of real programs.

Once the average working set size is known, the fault rate values under working set (WS) algorithm can be obtained. For the independent reference model, the WS fault rate is equal to the probability that a page hasn't been addressed in the last T references and that it will be addressed in the next reference summed over all pages, i.e.,

$$F_{WS}[T] = \sum_{i=1}^{n} (1-p_i)^T p_i$$

In Figures 10 and 11, the WS fault rate of the WATFIV program with two different page sizes, and the WS fault rate of APL and FFT programs are shown. In each figure, the WS fault rate probability of the corresponding AØ inversion model is shown by dotted lines. The horizontal axis is the average working set size and the vertical axis is the fault rate. The fit of the points obtained from the model to the points measured on the actual programs, basically reflects the results illustrated in Figures 8 and 9.

In Figure 9, we notice that the model somewhat overestimates the working set size of the APL program. An explanation for this behavior will follow in the next part.

8. Comparison with LRU Stack Model

We have defined the LRU stack model for the sequence of page references. This model is strongly bound to the observed LRU stack depth distribution of the programs. The long run fault rate of LRU stack model, under the LRU algorithm, converges to the LRU fault rate of the program upon which the model is based. This property is built into the LRU stack model by setting the stack depth distribution $\{d_i\}$, i=1,2,...,n of the model equal to the relative frequency of the observed stack distances generated by an actual program. It is interesting to investigate the behavior of LRU stack model under systems other than LRU.

Similar to our earlier set of experiments, the traces of several programs have been used to construct the empirical LRU stack distance distributions. In each case, an LRU stack distribution is used to construct the corresponding LRU stack model. In order to compare the optimal fault rate behavior of an actual program with the respective LRU stack model, the MIN algorithm is used for both of them. We note that since the observed LRU stack depth densities are not monotone decreasing values, we don't expect that LRU would be optimal for the model.

In Figure 12, the result of the experiments on the APL program using the MIN and LRU algorithms are shown. The solid lines represent the actual programs and the dotted lines represent data points from the corresponding LRU stack models. The LRU algorithm, as well as the MIN algorithm, were applied by a simulation run for the actual program and the model. Therefore, the discrepancy between the LRU fault rate curve of the model and the corresponding program

gives a significance measure of the sampling error in the simulation of the model. The more interesting information in this figure is, of course, the behavior of LRU stack model under the MIN algorithm. We note that the model gives a good prediction of the MIN fault rates of the actual program. Like the AØ inversion model, the good fits are especially notable for lower range of memory sizes.

In Figures 13 and 14, the average working set sizes and the WS fault rate of the APL program are compared with their respective LRU stack model values. If we inspect Figures 9 and 13, we notice that both the AØ inversion model and the LRU stack model give up to about a 10% overestimation of the actual average working set sizes of the APL program for most window sizes. We can give an explanation for this by taking a closer look at the distribution of working set sizes of the APL program. In Figure 15, a histogram of the observed working set sizes for window size T=4000 units for this program is plotted. In this plot, we can distinguish three major peaks. Although this is not a typical working set histogram, nevertheless programs sometimes do exhibit this behavior. Each peak can be associated with a large period of time which the program predominately spends in a locality which is different in size from other major localities. The frequent locality changes may also contribute to the clusters of fairly large working set sizes in the histogram.

Programs like APL which exhibit distinctive multiple locality regions give the illusion of being programs with fairly scattered reference patterns for the averaging mechanisms which build the models, e.g., AØ inversion and LRU stack models. The overestimation of the average working set sizes can be attributed to this averaging over the actual reference patterns.

Our other experiments show that the LRU stack model can predict reasonably well the MIN and WS fault rate of actual programs.

9. Extensions and Limitation of AØ Inversion Model

A possible extension of the model is in the area of management of filing systems. The files should be considered as variable size blocks of information. Therefore, we would have to introduce new parameters in the model which describe the file lengths.

The AØ inversion model can be easily extended to study the page read/write characteristics of the programs. The immediate application of such an extension would be the performance evaluation of the memory hierarchy systems with different page read/write transportation costs.

In finding the parameters of the AØ inversion model, we may encounter two kinds of problems. The first problem deals with solving the recurrence relations (3), and the second problem is related to the tail probabilities.

We recall that the MIN fault rates of an n page program are substituted for F_i's in (3) and, subsequently, the equations are solved for p_i's. It is theoretically quite probable that a set of F_i's, $1 \leq i \leq n$ and $F_i \geq F_j$ for $i < j$, are defined for which there is no real valued solution for p_i's. In fact, it is much harder to come up with some empirical values for F_i's where we can solve for p_i's.

The case where we can't solve the equations signifies the situation where there is no independent reference model with AØ fault rates exactly equal to those values that we have substituted for F_i's.

Our experiments in using the actual program traces show that for traces of reasonable length, we usually can find fairly accurate values for p_i's. However, when the measured MIN fault rate values are such that the equations (3) cannot be solved for all values of p_i's, we can find approximate values for these parameters by using the relations:

$$p_i = F_i - F_{i+1} . \qquad (5)$$

Once a p_i is found in this way, we can try to use relations (3) to find the successive parameters. For instance, in the FFT1 program, p_1 was found using (5) and the remaining probabilities were obtained by (3). The model seems to function properly even with approximate reference probabilities obtained from the above procedure.

The other problem is in finding the tail probabilities. Consider a program with n pages. Denote by F_m the fault rate of the program with memory size m under the MIN algorithm. When m becomes large, it is possible that for some memory size n' the observed F_i, i=n', n'+1,...,n will become zero. Here we assume that initial faults, due to the initial loading of the memory, are excluded from the total fault counts. Since F_m is the minimum fault rate with memory size m, then for any other fixed memory size paging algorithm the lower bound on the maximum memory size, n", for which it produces non-zero fault rate, is equal or greater than n'. For instance, for the WATFIV program, n'=120 and n"=164 (under LRU) and for the WATEX program, n'=n"=57 under MIN and LRU.

The point is that the AØ inversion method, which uses the MIN fault rate of the programs, can give us only n'-1 non-zero reference probabilities. Therefore, we get a model with n'-1 parameters and, clearly, when we use the model as it is, the pages n' through n never get referenced. For the lower range of memory sizes, the model with n'-1 parameter still gives satisfactory results. This is because, in the practical cases, the reference probabilities close to the tail of the model are very small. However, the behavior of the model can be greatly degraded for large memory sizes if we don't extend the tail probabilities to get a full size n parameter model.

Extending the tail probabilities to get n non-zero reference probabilities is still an open question here. We have chosen an ad-hoc method to get around the problem; we have simply extended the last non-zero reference probability so

that $p_{n'-1} = p_{n'} = \ldots = p_n$. Then we need to normalize to get a consistent set of probabilities. This solution has almost no effect on the performance of the model for small memory sizes, but it has greatly improved its performance in the region of large memory sizes.

10. Conclusion

Constructing program models can be a compact way of characterizing the page reference behavior of actual computer programs. In this paper, we have presented the technique of building an AØ inversion independent reference model, based on the actual MIN fault rates of a page reference trace. We noted that the independent reference model preserves the relative fault rate of actual program traces under MIN and LRU algorithms. Thus, the AØ inversion model should be capable of predicting the true LRU and FIFO fault rates of real programs for different main memory sizes. We presented the results of experiments on several programs to validate the model.

The AØ inversion model is also successful at predicting the average working set size and the WS fault rate of programs for a wide range of window sizes.

We have also seen that when an LRU stack model is constructed, based on the actual LRU distribution of a reference string, it can reasonably predict the MIN fault rate of the same program.

The analytical tractability and the simple probability structure of the AØ inversion model make this model a convenient tool for the analysis and evaluation of virtual memory systems and the performance of CPU's with high speed buffers.

When a program has several very distinctive locality regions, the AØ inversion model, as well as the LRU stack model, overestimates the average working set size by a small percentage. However, the prediction accuracy of the average fault rate under fixed memory size algorithms are virtually unaffected.

The problem of finding the tail probabilities has been dealt with here in an ad-hoc manner. More elaborate treatment of this subject should justify the desired accuracy of the model under very large memory sizes where the effect of these probabilities are most noticeable.

The independent reference assumption on the successive references of a program is against our intuition and the actual observations. However, we have demonstrated that by putting enough structure into the model, we can obtain a powerful model which produces realistic results, and can be used effectively in the analysis, simulation, and evaluation of several problem areas in memory management techniques.

BIBLIOGRAPHY

1. "The Cray-1 computer preliminary reference manual," CRAY Research, Inc.
2. "IBM System 360 Model 85 functional characteristics," Form A22-6916.
3. "Introduction to the IBM 3850 Mass Storage System (MSS)" Form GA32-0028-1.
4. Aho, A.V., Denning, P.J., Ullman, J.D., "Principles of optimal page replacement," J. of ACM 18, 1 (January 1971), pp 80-93.
5. Belady, L.A., "A study of replacement algorithms for virtual storage computer," IBM System J., 5, 2(1966), pp 79-101.
6. Coffman, E.G., Denning, P.J., "Operating system theory," Prentice Hall, Englewood Cliffs, New Jersey (1973).
7. Conti, C.J., "Concepts for buffer storage," Computer Group News (March 1969).
8. Denning, P.J., "Virtual memory," Computing Surveys, 2, 3, (1970).
9. Denning, P.J., "On modeling program behavior," AFIPS Conf. Proc., Spring Joint Computer Conference (1972), pp 937-944.
10. Denning, P.J., Savage, J.E., Spirn, J.R., "Models for locality in program behavior," TR 107, Computer Science Laboratory, Dept. of Electrical Engineering, Princeton University (1972).
11. Franaszek, P.A., Private communication (March 1974).
12. Franaszek, P.A., Wagner, T.J., "Some distribution free aspects of paging algorithm performance," J. of ACM, 21, 1 (January 1974).
13. King, W.F., "Analysis of paging algorithms," IBM Watson Research Center Research Report RC-3288 (March 1971).
14. Mattson, R.L., Gecsei, D.R., Traiger, I.L., "Evaluation techniques for storage hierarchies," IBM Systems Journal, 9, 2 (1970).
15. Rafii, A. "Empirical and analytical studies of program reference behavior," Ph.D. Dissertation, Electrical Engineering Dept., Stanford University: SLAC Report 197, Stanford Linear Accelerator Center (July 1976).

16. Shedler, G.S., Tung, C., "Locality in page reference strings," SIAM Journal of Computing 1, 3 (September 1972).

17. Watson, R.W., "Time sharing system design concepts," McGraw-Hill, New York (1970).

Figure 1: PAGE FAULT RATE OF WATFIV PROGRAM

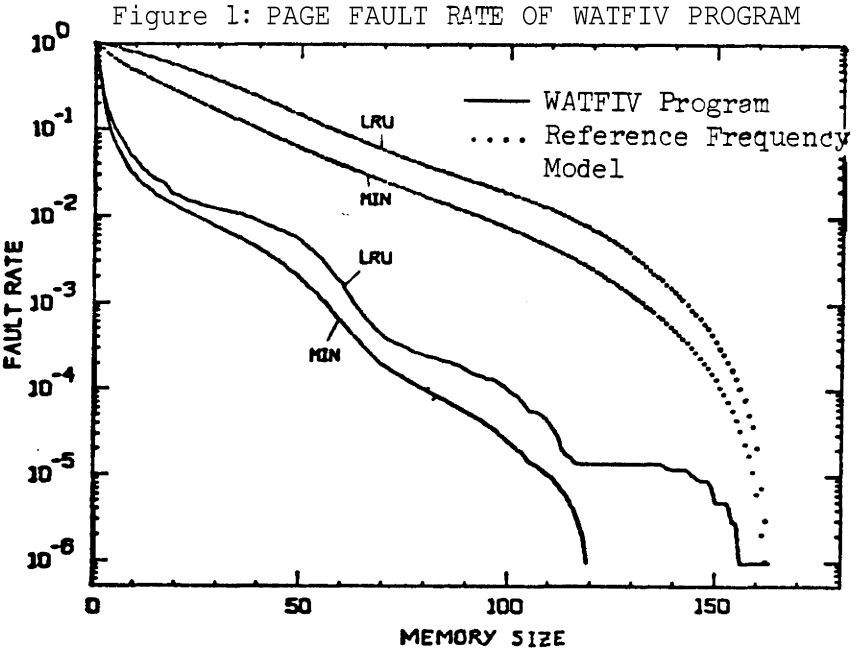


Figure 2: PAGE FAULT RATE WATFIV ϵ A# MODEL

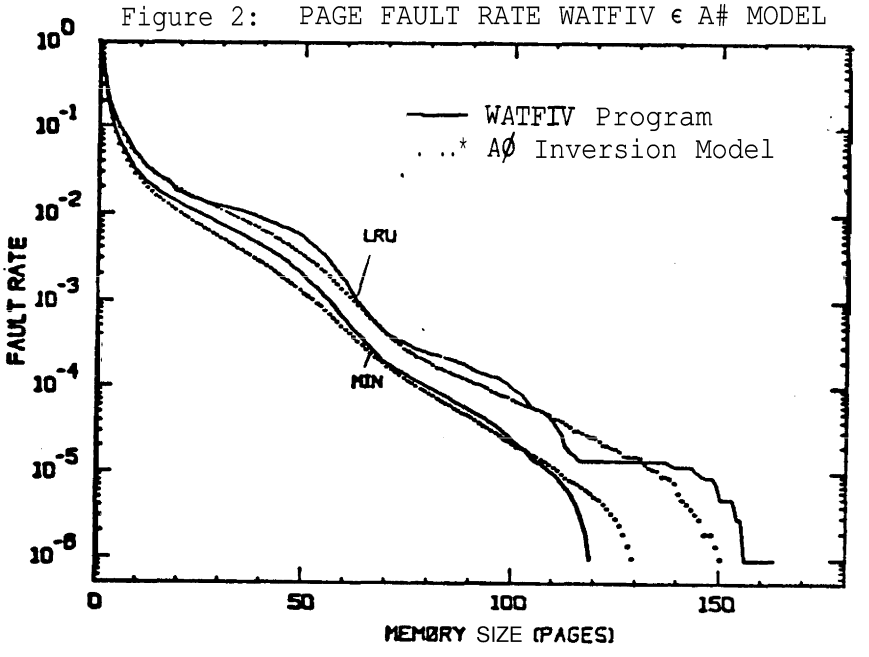


Figure 3: REFERENCE PROBABILITIES FOR TWO MODELS

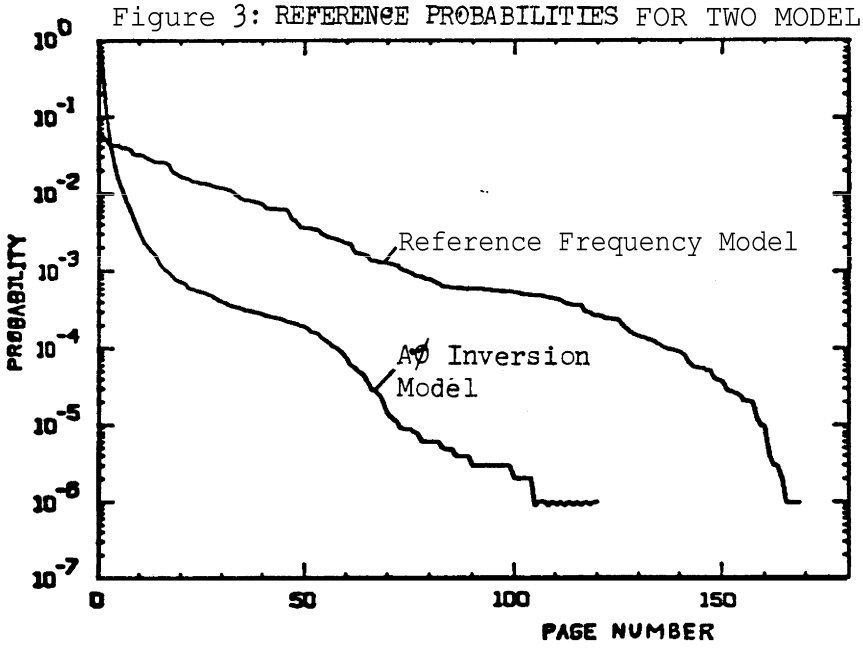


Figure 4: FIFO FAULT RATE OF WATFIV ϵ AØ MODEL

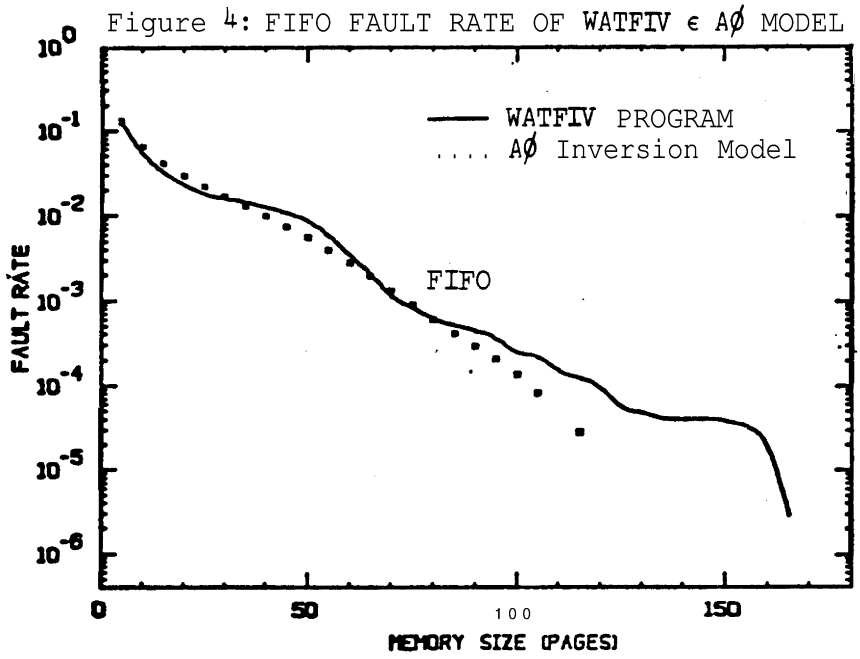


Figure 5: LRU ϵ MIN PAGE FAULT RATES

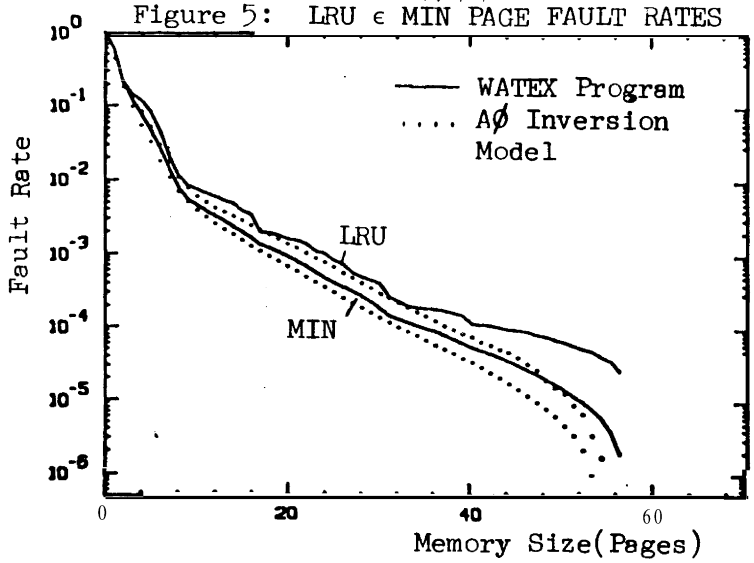


Figure 6: LRU ϵ MIN PAGE FAULT RATES

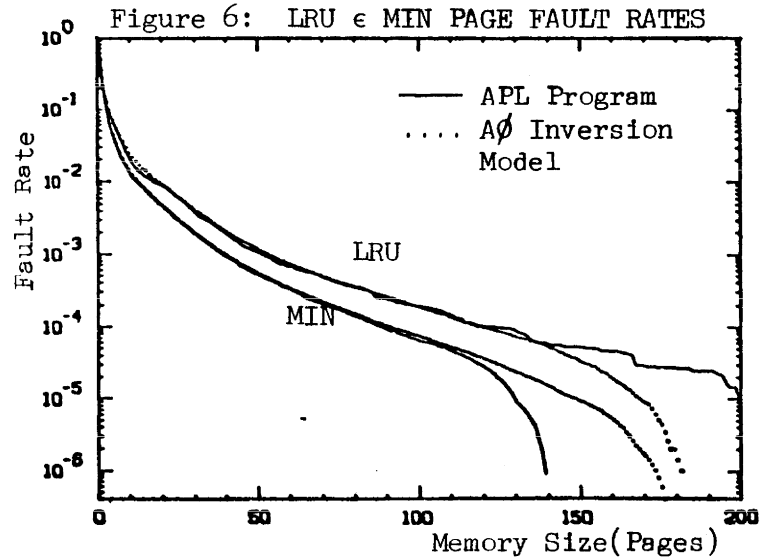


Figure 7: LRU ϵ MIN PAGE FAULT RATES

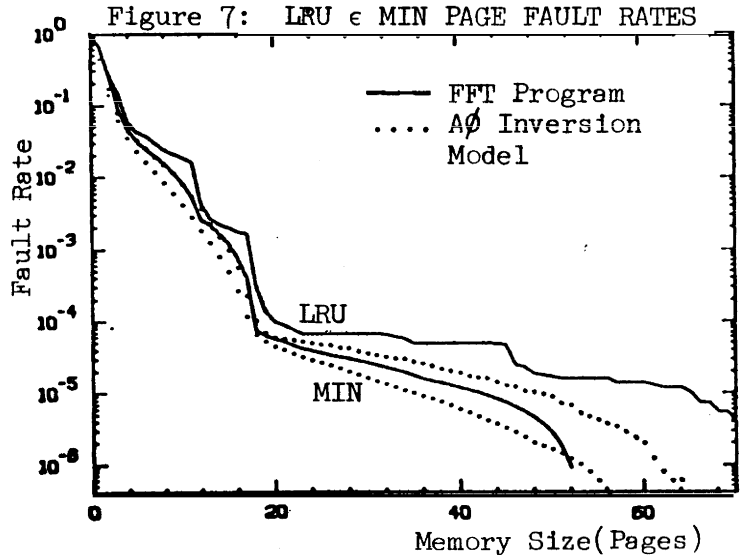


Figure 8: AVERAGE WORKING SET SIZES

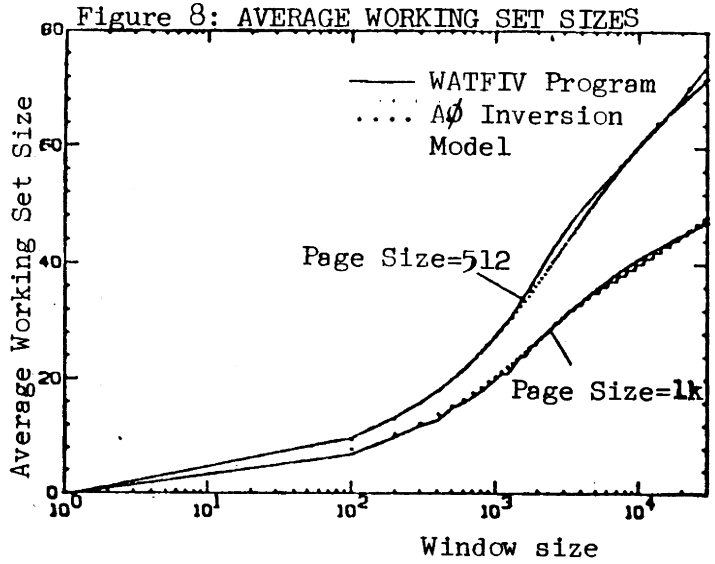


Figure 9: AVERAGE WORKING SET SIZES

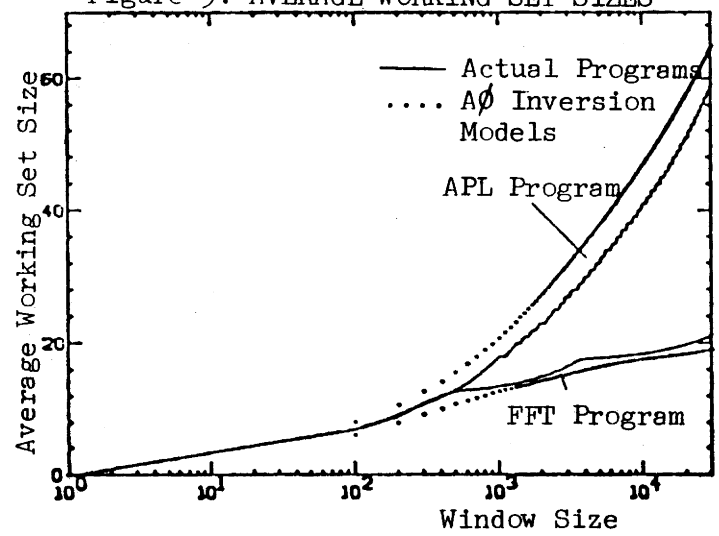


Figure 10: WS PAGE FAULT RATES

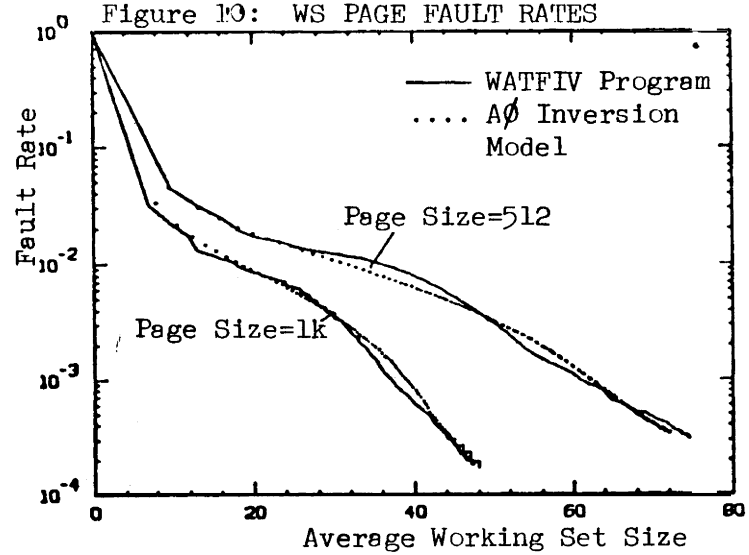


Figure 11: WS PAGE FAULT RATES

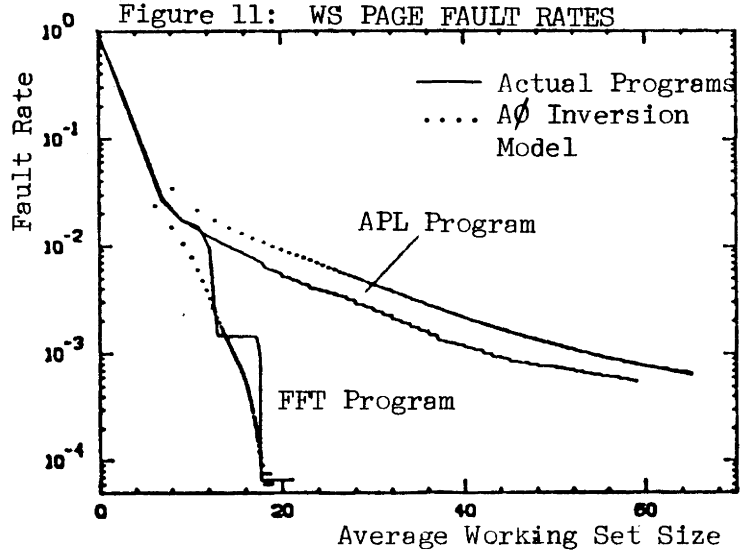


Figure 12: FAULT RATE OF APL ϵ LRU STACK MODEL

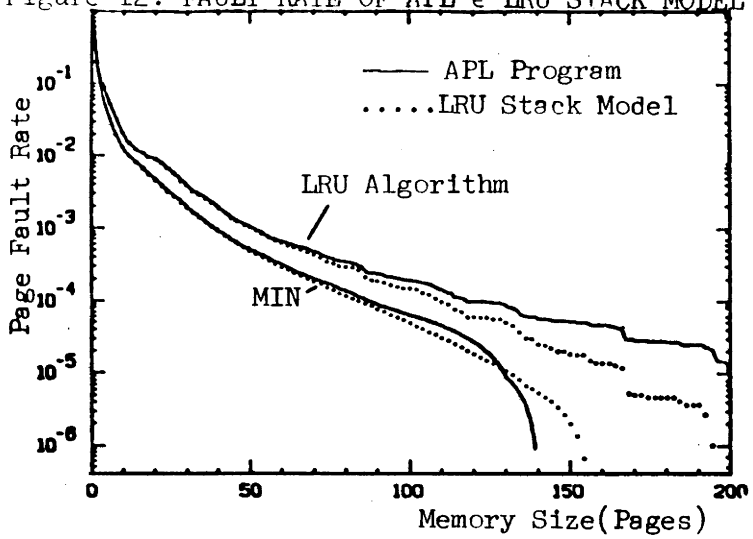


Figure 13: AVERAGE WORKING SET SIZES

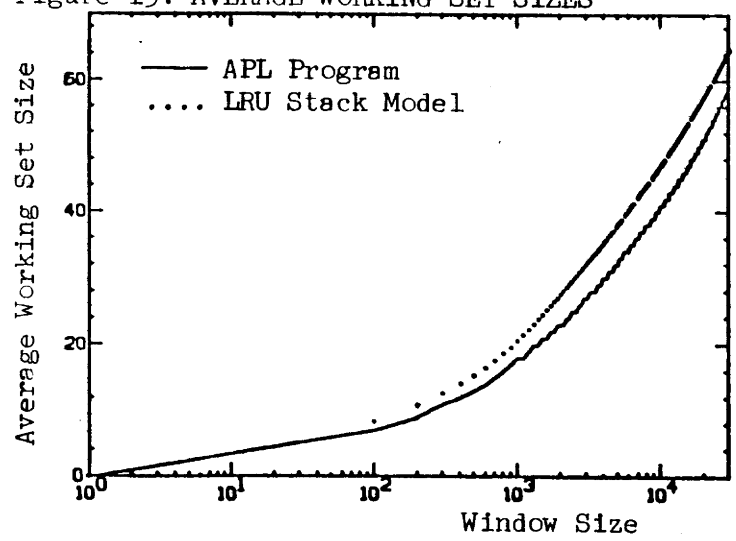


Figure 14: WS PAGE FAULT RATES

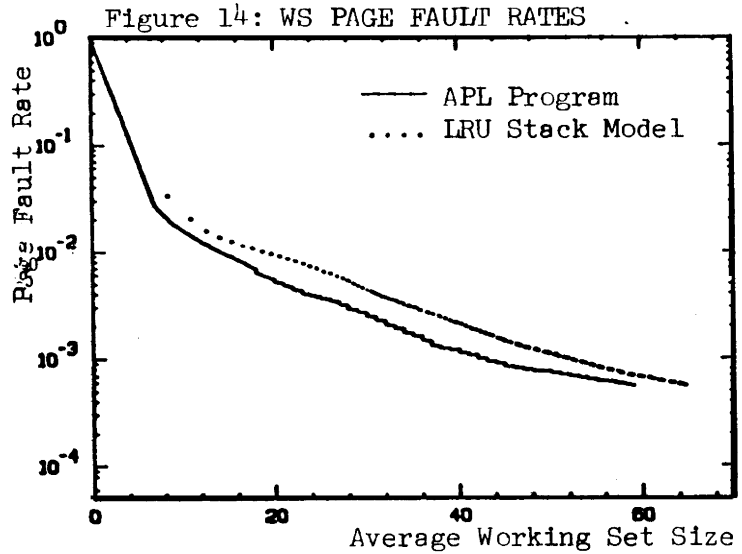


Figure 15: HISTOGRAM OF APL WORKING SET SIZES