SIAM J. COMPUT.
Vol. 5, No. 4, December 1976

ANALYSIS OF THE OPTIMAL, LOOK-AHEAD DEMAND PAGING ALGORITHMS*

ALAN JAY SMITH†

Abstract. We express the future behavior of programs that may be described by two common program behavior models, the independent reference model and the LRU stack model, by a discrete time Markov chain. Using this Markov chain model, we are able to calculate the theoretical minimum number of page faults for a program representable by either of these models in either a fixed or variable size memory. The behavior of optimal look-ahead and optimal realizable demand paging algorithms are compared, and it is seen that look-ahead paging demonstrates an inherent advantage sufficient to account for the differences observed between currently implemented demand paging algorithms and theoretically optimal algorithms.

Key words. paging, virtual memory, memory management, LRU stack model, independent reference model, Markov chains, MIN algorithm

1. Introduction. A paged *virtual memory*, implemented as early as 1958 in the Atlas computer (Kilburn et al. (1962)), has become a common feature of large modern operating systems. Memory in such systems is divided into fixed size blocks called *page frames* and the address space of each process is divided into *pages*. The memory consists of at least two levels; a four or five level memory hierarchy including fast buffer storage (cache), main memory (core), drums, disks and tapes is common. If the pages are transferred from secondary storage to main storage only when needed, then the process is called *demand paging*. Every time an attempt is made to access a page which is not in main memory, a *page fault* is said to occur and this page is brought into memory. A page fault is costly in terms of lost processing time in two ways: the processor must execute the *page replacement algorithm* which removes a page from memory and thereby frees a page frame for the incoming page and it must execute the *page fetch algorithm* which finds the incoming page and initiates the transfer. The processor may also have to wait, if there is no other process ready to run, while that page is fetched. Effective operation of a computer system with virtual memory therefore requires that the amount of processor time wasted due to page faults be minimized.

The choice of a good page replacement algorithm is very important to minimizing the number of page faults. A poor replacement algorithm would choose pages likely to be needed again in the near future, thus precipitating further page faults. It can be shown that given complete knowledge of the future behavior of the program, the optimal paging algorithm for fixed memory size, MIN (the one that causes the minimum number of page faults), will remove that page among those in memory that will be referenced furthest in the future (Mattson et al. (1970), Pomeranz (1971)). Belady presents an algorithm for

* Received by the editors April 8, 1975, and in final revised form December 2, 1975.

† Computer Science Division, Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, University of California at Berkeley, Berkeley, California 94720. This research was supported in part by the National Science Foundation under Grant DCR 74-18375, and in part by the University of California. Computer time was provided by the Energy Resources Development Administration under Contract AT(04-3)-515.

calculating this minimum number (Belady (1966)) in one pass over a program trace tape and he extends it to calculating the minimum number of faults for a number of different memory sizes at once in Belady and Palermo (1974). Mattson et al. (1970) present an algorithm for calculating the minimum number of faults (they call their algorithm OPT) in two passes over the trace tape. Lewis and Nelson (1974) also present algorithms for calculating this minimum number of faults. We show below that for certain models of program behavior it is possible to estimate analytically the number of faults generated by the MIN algorithm.

A number of paging algorithms have been proposed, such as working set (Denning (1968)) and page fault frequency (Chu and Opderbeck (1972)) that are designed to vary the number of page frames allocated to a process as its memory needs change. These algorithms assume that the computer system is multiprogrammed and that page frames may be transferred between processes. Prieve and Fabry (1975) present an algorithm which yields the optimal variable space memory allocation for a process when measured in virtual time. That is, this algorithm minimizes the number of page faults for any given average memory size when that average size is calculated during the period the process is active. In § 4 of this paper we show that it is also possible to estimate the behavior of this optimal algorithm, VMIN, analytically.

A model for program behavior that has been proposed for its convenience of analysis is the *independent reference model* (IRM) in which the probability of referencing page i at time t is p_i for all t. Baskett and Rafii (1975) have shown that the proper choice of the reference probabilities allows a number of program characteristics to be accurately captured by this model. Aho, Denning and Ullman (1971) demonstrate an algorithm A_0 which they prove is the optimal demand paging algorithm for the independent reference model. This algorithm keeps in memory the $M-1$ pages, for memory size M, with the largest values of p_i. The remaining page frame is used for other pages that are referenced. King (1971) calculates the fault rate for the independent reference model when using the least recently used (LRU), first in, first out (FIFO) or A_0 paging algorithms. Franaczek and Wagner (1974), Coffman and Denning (1973) and Denning and Schwartz (1972) also consider the independent reference model.

Another model for program behavior which has a number of useful features is the LRU *stack model*, which is discussed by Coffman and Denning (1973), Denning, Savage and Spirn (1972) and Oden and Shedler (1972). In this model, the sequence of LRU stack distances, $D=\{d_i\}$, (Mattson et al. (1970)) are a sequence of independent, identically distributed random variables. This model provides for locality of reference (Denning (1972)), but it fails to represent changes in the size of the locality or abrupt changes in the content of the locality. Lewis and Yue (1971) and Lewis and Shedler (1973) reject the LRU stack model using formal statistical techniques, but as we show in Fig. 5 (see § 5), there is reason to believe it to be a good approximation. If q_i is the probability of a hit to level i of the stack, and if $q_i \geqq q_j$ for $i \leqq j$, then it can be shown that LRU is the optimal demand paging algorithm (Coffman and Denning (1973)).

In the next two sections of this paper we show that it is possible to calculate analytically the behavior of the MIN algorithm when run on a trace of a program obeying either the LRU stack model or the independent reference model. In § 4

we analyze the behavior of the VMIN algorithm when applied to a program that can be described by either the independent reference model or the LRU stack model. Because MIN and VMIN minimize the fault rate for a specific reference string, they have an inherent advantage over realizable algorithms, even optimal ones such as A_0, which only minimizes the fault rate on the average over all IRM reference strings. Measurements and calculations to be presented in § 5 (some of which also appear in Baskett and Rafii (1975)) indicate that the observed difference between optimal (look-ahead) algorithms and realizable ones when run on real program traces is similar to the difference found when such a comparison is made on traces generated from independent reference model or LRU stack model simulations. In § 6, the conclusion, we indicate the applicability of our results.

2. The independent reference model. Let us consider a *program* B which consists of M *pages* using a memory of N *page frames* $(N<M)$. The *reference string* $R=(r_1, r_2, \cdots, r_i, \cdots)$ is the sequence of memory references, listed in the order in which they occur. We need know only which page is referenced; thus r_i will denote a page number, $1 \leqq r_i \leqq M$. The independent reference model assumes that the reference string R has the following property: $P[r_i = j] = q_j$ for all i. We assume without loss of generality that $q_j > 0$, $1 \leqq j \leqq M$.

We define the *state S of the process* B as (F, W) where F is the (unordered) set of pages currently in memory and W is the set of all pages in the program B listed in the order of their first future occurrence. It may be seen that the reference string R is independent of the state of the memory F; thus the sequence of states $\langle W_i \rangle$ for the future behavior of the process is independent of the memory state also. Except where it will cause confusion, we will use the word "state" interchangeably to refer to the state of the entire system, (F, W), the future reference state W and the memory state F. We shall also omit super- and subscripts where it causes no confusion.

A future reference state W is denoted by a permutation of the numbers $1, \cdots, M$. The state $W_t = (w_1^t, w_2^t, \cdots, w_M^t)$ at time t is defined by: $w_1^t = r_{t+1}$, and for $j>1$, $w_j^t = \min_k (r_k,\ k>t$, such that $r_k \notin (w_1^t \cdots w_{j-1}^t))$. The steady state probability of state W is

$$P[W] = \prod_{k=1}^{M} \frac{q_{w_k}}{\sum_{j=k}^{M} q_{w_j}} \tag{1}$$

which may be shown (Mattson et al. (1970)) to be the same as the steady state probabilities for the LRU stack, as derived by King (1971) and by Coffman and Denning (1973). This formula may be quickly obtained by noting that the probability that $w_k = j$ is simply q_j, the probability that page j is referenced at an arbitrary time t, normalized by the "remaining reference probability" to be accounted for, $1 - \sum_{j=1}^{k-1} q_{w_j}$ or $\sum_{j=k}^{M} q_{w_j}$.

The set of possible successors $\{W_{t+1}\}$ to W_t, for $W_t = (w_1^t, \cdots, w_M^t)$, is

$$\{(w_1^t, w_2^t, w_3^t, \cdots, w_M^t), (w_2^t, w_1^t, w_3^t, \cdots, w_M^t), (w_2^t, w_3^t, w_1^t, w_4^t, \cdots, w_M^t), \cdots$$
$$(w_2^t, w_3^t, \cdots, w_1^t, w_M^t), (w_2^t, w_3^t, \cdots, w_M^t, w_1^t)\}. \tag{2}$$

For all W_t,

$$P[W_{t+1} = W^* | W_t] = \begin{cases} \dfrac{P[W^*]}{\sum_{\forall W' \in \{W_{t+1}\}} P[W']} & \text{if } W^* \in \{W_{t+1}\}, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $P[W^*]$ and $P[W']$ are defined in (1) and $\{W_{t+1}\}$ in (2). Equation (3) follows from the simple probability relation $P(A|B) = P(AB)/P(B)$ where A is the event that page w_1^t becomes the kth element in the future references state, B is the event that the remainder of the elements are ordered as $(w_2^t, w_3^t, \cdots, w_M^t)$ and AB is the occurrence of both of these events.

Let $S_t = (F_t, W_t)$ be the state of the process at the time of the tth reference. Define

$$p_{i,j} = p\{S_{t+1} = j | S_t = i\} \quad \text{for all } t; \tag{4}$$

that is, $p_{i,j}$ is the state transition probability. Let $p_{i,j}^w$ be the state transition probabilities for the future reference state, W, as defined in (3).

We define the *page fault rate*, PFR, as

$$\text{PFR} = \sum_{\forall S_i \text{ such that } w_1 \notin F_i} P[S_i]. \tag{5}$$

We show below that we can calculate the steady state page fault rate given the state transition probabilities.

The state transition probabilities depend on the paging algorithm. We define a paging algorithm as follows:

Case 1. $r_{t+1} \in F_t$. Then no page fault occurs and $F_{t+1} = F_t$.

Case 2. $r_{t+1} \notin F_t$. Then a page fault must occur. Let $F = (f_1, \cdots, f_N)$. Let $f_v = \max_k (w_k \text{ such that } \exists f_i = w_k)$. That is, f_v is that page in memory that will not be referenced for the longest time. The memory state changes as follows:

$$F_{t+1} = F_t - \{f_v^t\} + \{r_{t+1}\}. \tag{6}$$

The paging algorithm we have defined is identical to Belady's (1966) MIN algorithm, and it has the property that it is the demand paging algorithm that causes the minimum number of faults. We note also that our choice of a state space is such that we are able to specify the behavior of the system when the MIN paging algorithm is used; a paging algorithm such as LRU would require not the future state of the system but that the past state of the system be specified (King (1971)).

We may combine the transition information given above for the memory state F and the future reference state W to get the transition probabilities p_{ij} and matrix $P = \{p_{ij}\}$ for the state of the system.

$$p_{ij} = \begin{cases} p_{ij}^w & \text{if } w_1 \in F_i, F_j = F_i, W_j \in \{W_i'\} \text{ or} \\ & \text{if } w_1 \notin F_i, F_j = F_i - \{f_v\} + \{w_1\}, \\ & \text{and } W_j \in \{W_i'\}, \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where f_v is defined in (6) and $\{W_i'\}$ is the set of successor future reference states to W_i. These state transition probabilities are immediate from the memory transition specification in (6) and the future reference state transition probabilities in (3).

Let $S_j = ((f_1 \cdots f_N), (w_1 \cdots w_M))$ and let S_i be some arbitrary other state. Then the following sequence of page references will always serve to take us from state S_i through state S_j:

$$f_1 \cdots f_N, f_1 \cdots f_N, w_1 \cdots w_M; \tag{8}$$

that is, we will be in state S_j immediately after the second reference to f_N in the string given above. We may also enter state S_j by the following reference string:

$$f_1 \cdots f_N, f_1 \cdots f_N f_N, w_1 \cdots w_M, \tag{9}$$

where we enter S_j after the third reference to f_N. From our definition of the transition probabilities in (7), it is evident that the state transitions depend only on the immediately prior state. It can thus be seen that we have defined an irreducible, nonperiodic, finite state Markov chain.

It was shown in the paragraph above that any state S_j is reachable from any other state S_i. The number of possible states then is the number of possible future states, times the number of possible memory states, or:

$$M! \cdot \binom{M}{N}. \tag{10}$$

The steady state probabilities may be found as the solution of

$$\pi = \pi P, \tag{11}$$

where π is the vector of steady state probabilities, i.e., $\pi_i = P\{S_i\}$. The elements of the matrix P were defined in (7). The calculation for the steady state probabilities thus involves the solution of a set of equations whose number grows as the expression in (10), which grows more quickly than factorially with the number of pages in the program.

3. The LRU stack model. We consider, as in the previous section, a program B which consists of M pages using a memory of N page frames ($N < M$). The reference string R is the sequence of page numbers in the order that program B references them. Let $D = (d_1, d_2, \cdots, d_i, \cdots)$ be the LRU distance string (Mattson et al. (1970)). That is, a distance of d_t at reference t (to page r_t) means that page r_t was, immediately prior to reference t, the d_tth most recently referenced page. The LRU stack model assumes that the distance string has the following property: $P[d_t = j] = q_j$ for all t. We assume $q_j > 0$, $j = 1, \cdots, M$.

The state S of the process B may be defined again as (F, W) where F is the unordered set of pages currently in memory and W is the set of all pages in program B, listed in the order of their first future occurrence. We note that the LRU stack model for program behavior is a probabilistic description that is independent of the specific identities of the pages. Coffman and Denning (1973) show, as might be expected, that the equilibrium probability of a reference to page i is $1/M$ for all i and that therefore the mean time between referencing a page is M.

We let the future reference state W_t be $(w_1^t \cdots w_M^t)$ as in § 2. The set of successors to W_t, $\{W_t'\} = \{W_{t+1}\}$, is as defined in (2). Mattson et al. (1970) show that each LRU stack hit at distance d_i (at time $t+u$) is also a future stack hit at distance d_i (at time t) where t and $t+u$ are the times of successive references to the given page, so the probability of a page that has just been referenced appearing next in the future stack at distance j is q_j. Thus for all W_t and all t.

$$P[W_{t+1} = (w_2^t \cdots w_j^t w_1^t \cdots w_M) | W_t = (w_1^t \cdots w_M^t)] = q_j. \tag{12}$$

It is also possible to obtain this result by direct calculation, as the interested reader may easily discover. Let $(w_2 \cdots w_j w_1 \cdots w_M)$ be defined as the jth successor of $(w_1 \cdots w_M)$.

It is possible to greatly reduce the state space for this process. Let $G_t = (g_1^t \cdots g_N^t)$ be the set of pages in main memory at time t identified by their position in the future stack. For example, if the future stack contents are $W_t = (4^*, 1, 5^*, 2^*, 6, 3)$, where $M = 6$, $N = 3$, and $F_t = (4, 5, 2)$, then $G_t = (1, 3, 4)$. The "*" has been used to denote those pages in main memory and thus the elements of G_t. The updating of both the future stack and the memory state is independent of the specific identities of the pages, so that this state space reduction has not eliminated any useful information.

We now let $S_t = (G_t)$ be the state of the process at the time t. Let p_{ij} be the state transition probabilities, which are defined as

$$p_{ij} = P[S_{t+1} = j | S_t = i] \quad \text{for all } t. \tag{13}$$

We define the page fault rate, PFR, as

$$\text{PFR} = \sum_{\forall S_i \text{ such that } 1 \notin G_i} P\{S_i\}. \tag{14}$$

The paging algorithm we will employ is defined as follows:

Case 1. $1 \in G_t$. Then no page fault occurs. Let the page r_t next appear at a distance of k in the future stack. Then $G_{t+1} = (g_1^{t+1}, g_2^{t+1}, \cdots, g_N^{t+1})$, where

$$g_i^{t+1} = \begin{cases} g_i^t & \text{if } k < g_i^t, \\ g_i^t - 1 & \text{if } k \geq g_i^t \text{ and } g_i^t \neq 1, \\ k & \text{if } g_i^t = 1. \end{cases}$$

In this case, let G_{t+1} be the kth successor of G_t.

Case 2. $1 \notin G_t$. A page fault occurs. Let r_t and k be as in Case 1. Let $j = \max_i (g_i^t)$. Let $G_t^0 = G_t - \{j\} + \{1\}$. Then the successors of state G_t are the same as the successors of state G_t^0 and occur with the same probabilities. The kth successor of G_t is defined as the kth successor of G_t^0.

We have, as in § 2, defined an algorithm that is identical to Belady's (1966) MIN algorithm; i.e., we have always chosen that page to remove from main memory that will not be referenced for the longest time.

The transition probabilities for the state G_t for all t are defined as:

$$p_{ij} = \begin{cases} q_k & \text{if } G_j \text{ is the } k\text{th successor of } G_i, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

It was shown in § 2 that there exists a sequence of page references that will create any arbitrary state of pages in memory and any arbitrary future state. Since $q_k > 0$, for all k, it follows, therefore, that any state G_i is reachable from any state G_j in t or $t+1$ steps, for sufficiently large (but finite) t. By the same reasoning used earlier, we see that we have defined a discrete-time, finite state, nonperiodic, irreducible Markov chain. The equilibrium state probabilities, π_i, may be obtained as indicated in (11). The number of states is $\binom{M}{N}$, which is $M!/(N!(M-N)!)$. For a constant N, the size of the memory, the number of states grows factorially with the number of pages in the program, although not nearly as fast as the number of states in the independent reference model.

4. The VMIN algorithm.

4.1. The independent reference model. Let q_i be the probability of reference to page i, as defined in § 2. The time t between references to page i is distributed geometrically as

$$q_i(1-q_i)^{t-1} \tag{16}$$

with mean time between references

$$1/q_i. \tag{17}$$

The probability that the time between references is greater than τ (where τ is the working set parameter (Denning (1968)) is

$$(1-q_i)^{\tau}. \tag{18}$$

Because the time between references is geometrically distributed, the backward recurrence distance (Cox(1962)) is also geometrically distributed with the same mean. Therefore the probability that a given page i has not been referenced more recently than the preceding τ time units is also $(1-q_i)^{\tau}$ as in (18). This latter probability is just the probability that a reference to page i will result in a page fault when using the working set paging algorithm with working set parameter τ. Therefore the fault rate when using the working set algorithm on a program described by the independent reference model is

$$\sum_{i=1}^{M} q_i(1-q_i)^{\tau}. \tag{19}$$

The average memory space occupied for working set parameter τ is simply the sum of the probabilities, summed over the pages, that page i is in memory. Page i is in memory if it has been referenced in the preceding τ time units, so the average memory space used is

$$\sum_{i=1}^{M} (1-(1-q_i)^{\tau}). \tag{20}$$

The results above in this section have all been presented by Denning and Schwartz (1972) and Baskett and Rafii (1975).

The VMIN algorithm (Prieve and Fabry (1975)) is that algorithm that minimizes the number of page faults for given virtual (process) time average

memory use. It works as follows: for a fixed τ, remove all resident pages that will not be referenced in the following τ time units. By varying τ, a (piecewise linear) curve is traced out in the plane of page faults vs. average memory size. We note that since VMIN is a demand algorithm, it cannot behave symmetrically with working set. The time average probability that a page is in memory is the time average probability that the interreference interval is of length τ or less. This is the ratio of the probability that the interreference interval is $\leqq \tau$ times the mean length of such an interval to the mean length of all intervals. Thus the average memory size for parameter τ when using the VMIN algorithm is:

$$\sum_{i=1}^{M} \frac{(1-(1-q_i)^{\tau})\left(\sum_{t=1}^{\tau} \frac{q_i(1-q_i)^{t-1}t}{(1-(1-q_i)^{\tau})}\right)}{\sum_{t=1}^{\infty} q_i(1-q_i)^{t-1}t} = \sum_{i=1}^{M} \frac{\sum_{t=1}^{\tau} q_i(1-q_i)^{t-1}t}{\sum_{t=1}^{\infty} q_i(1-q_i)^{t-1}t}$$

$$= \sum_{i=1}^{M} q_i^2 \sum_{t=1}^{\tau} (1-q_i)^{t-1}t \tag{21}$$

$$= \sum_{i=1}^{M} q_i^2\left[\frac{1-\tau(1-q_i)^{\tau}}{q_i} + \frac{(1-q_i)(1-(1-q_i)^{\tau-1})}{(1-(1-q_i))^2}\right]$$

$$= M - \sum_{i=1}^{M} (1-q_i)^{\tau}(1+\tau q_i)$$

(which, we observe, is less than for working set, as given in (20)). The page fault rate is given by (19); therefore we have calculated the expected fault rate and average memory allocation for the VMIN algorithm when using a given value of τ and when the program in question obeys the independent reference model.

4.2. The LRU stack model. Let q_i be the probability of a hit to level i in the stack, as defined in § 3. As discussed earlier (equation (12)), q_i is also the probability that the page just referenced will next be referenced when it is at level i in the stack. A page next referenced at level i will start at level 1 in the stack and successively occupy levels $1, 2, \cdots, i$ before it is referenced.

Let $\bar{Q}_i = \sum_{j=i+1}^{M} q_j$, and let $f_i(j)$ be the probability mass function (pmf) for the duration of time a page spends at level i in the stack, given that it is not referenced at level i in the stack. Then $f_i(j)$ is geometrically distributed, since all references are independent and follow the LRU stack model, and it is equal to

$$f_i(j) = \frac{\bar{Q}_i}{1-q_i}\left(1-\frac{\bar{Q}_i}{1-q_i}\right)^{j-1}. \tag{22}$$

This expression (equation (22)) holds even for $i = 1$ if we define $0^0 \triangleq 1$.

Let $g_k(j)$ be the probability mass function for the time a page spends at level k in the stack, given that it will be referenced at level k. Then

$$g_k(j) = \frac{q_k}{1-\bar{Q}_k}\left(1-\frac{q_k}{1-\bar{Q}_k}\right)^{j-1}. \tag{23}$$

The probability mass function for the time until a page is referenced, given that it is next referenced at level k, is

$$h_k(j) = \left(* \prod_{i=1}^{k-1} f_i(j) \right) * g_k(j), \tag{24}$$

where "$*$" denotes convolution and "$* \Pi$" means the convolution product. Summing over all k, we have as the pmf for the time between references to a page:

$$e(j) = \sum_{k=1}^{M} h_k(j) q_k \tag{25}$$

and for the cumulative function,

$$E(j) = \sum_{i=1}^{j} e(i). \tag{26}$$

The pmf for the time to the last reference to an arbitrary page at an arbitrary time is simply the backward recurrence time (Cox (1962, p. 61)) and is equal to

$$M(1-E(t)). \tag{27}$$

The probability that an arbitrary page is in core is simply the probability that the backward recurrence time is less than or equal to τ, the working set parameter; thus we have

$$M \sum_{t=1}^{\tau} (1-E(t)) \tag{28}$$

for the probability that an arbitrary page is in core. The mean number of pages in core is M times the value of (28) or

$$M^2 \sum_{t=1}^{\tau} (1-E(t)). \tag{29}$$

The fault probability for a parameter τ is

$$1-E(\tau), \tag{30}$$

the probability that a page has not been referenced in the preceding τ time units. Equations (29) and (30) trace out a fault rate/memory size curve that describes the behavior of the working set algorithm when executed on a program obeying the LRU stack model. There seems to be no simple closed form for these equations although the generating function for $e(t)$ is easily obtained. (In Denning et al. (1972) a recurrence relation is derived which permits computing $e(t)$ directly in t steps.)

As in (21), we can obtain an expression for the average memory usage for the VMIN algorithm when using parameter τ. The average memory space used is

$$M\frac{\sum_{t=1}^{\tau} e(t)t}{\sum_{t=1}^{\infty} e(t)t} = \sum_{t=1}^{\tau} e(t)t. \tag{31}$$

5. Calculations and simulations. In this section we present calculations of the MIN fault rate for two simple examples of the independent reference model and LRU stack model, and then we use simulation to compare the behavior of the algorithms discussed in this paper for two larger and more interesting cases. Exact numerical calculation appears limited, because of the combinatorial growth of the number of states, to systems with a dozen page frames or less.

Example 1. Consider a three page program obeying the LRU stack model, where $q_1 = q_2 = q_3 = \frac{1}{3}$. We note that a program described by the independent reference model with the same uniform values of $\{q_i\}$ will have exactly the same behavior. The state space (where the state G_t is given in the circles (see § 3)) and translation probabilities are diagrammed in Fig. 1, for a memory size of two page frames. The equilibrium state probabilities are: $p_{1,2} = \frac{4}{9}$, $p_{1,3} = \frac{1}{3}$ and $p_{2,3} = \frac{2}{9}$. The probability of a page fault occurring is then just $p_{2,3}$ or $\frac{2}{9}$. The probability of a page fault using any realizable demand paging algorithm is $\frac{1}{3}$, so that we observe that the optimal look-ahead algorithm has a 33% advantage over any realizable algorithm.

Example 2. We consider a program following either the independent reference model or the LRU stack model with $q_1 = q_2 = q_3 = q_4 = .25$. Let there be three page frames in primary memory. The state space and transition probabilities appear in Fig. 2. After calculation, we obtain as the steady state probabilities: $p_{1,2,3} = \frac{9}{22}$, $p_{1,2,4} = \frac{6}{22}$, $p_{1,3,4} = \frac{4}{22}$ and $p_{2,3,4} = \frac{3}{22}$. The fault probability is then $\frac{3}{22}$ (or .13636) which is much less than the fault rate of .25 to be expected from any realizable algorithm.

Because of the rapid growth in the number of states for the analysis of the MIN algorithm and the complexity of the analysis of the VMIN algorithm for the LRU stack model, and also because of the availability of an already written simulation program, we will present simulation results comparing these and other algorithms. We note that the complexity of our results does not negate their usefulness; rather it provides an incentive to develop a useful and simple approximation. In much the same manner that King's (1971) work provided the basis for

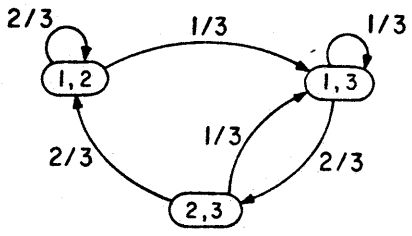

FIG. 1

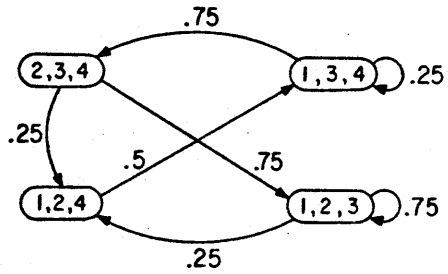

FIG. 2

some of the work leading to Baskett and Rafii's (1975) "A_0 inversion model", we believe that the existence of an exact solution to the fault behavior of nonrealizable algorithms will permit the development of more tractable approximations.

The values chosen for $\{q_i\}$ for our simulation of both the independent reference model and the LRU stack model are:

$$q_i = 2^{-i}, \qquad 1 \leqq i \leqq 13,$$
$$q_i = 2^{-13}, \qquad i = 14,$$
$$q_i = 0, \qquad i > 14.$$

The simulations were run for one million references each.

For the independent reference model, the A_0, MIN, VMIN, working set (WS) and LRU algorithms were employed, and the fault rate is shown in Fig. 3 for each of these algorithms. As might be expected, the A_0 algorithm demonstrated the best performance of any of the demand algorithms, although MIN, which is unrealizable, did better.

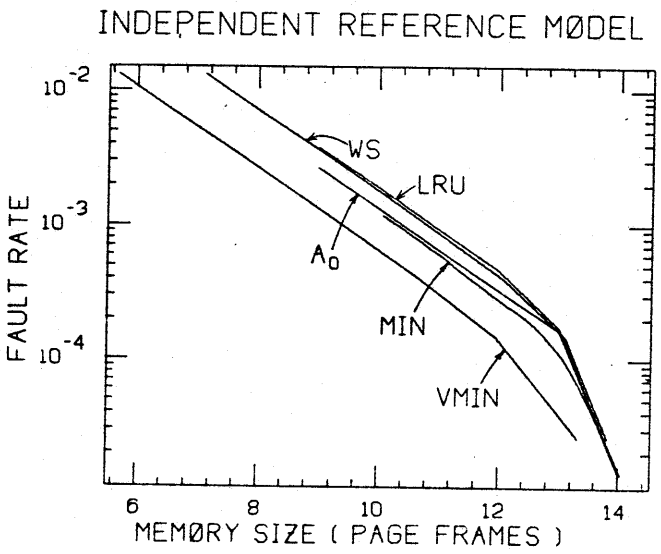

FIG. 3

The author conjectures that the observed result that working set performed better than LRU in this case is a general result for the independent reference model, although the author is not aware of any rigorous proof of this. (A "proof" by Denning and Schwarz (1972) was withdrawn by Denning (1973)). We do present the following heuristic argument; a complete proof (which we do not have) is, in any case, beyond the scope of this paper. We observe that, for the independent reference model, the reference pattern to any one page is completely independent of that to all other pages; thus each page may be studied in isolation for replacement. For both working set and LRU, a reference to a page constitutes a renewal epoch; that is, in both cases, replacement decisions for a given page are completely independent of the reference pattern preceding its last reference. Because of the independent reference patterns, the only measure of the expected time to the next reference to a page, given the renewal property described, is the *time* since the last reference, which is exactly what working set measures. LRU records the number of other pages referenced since the given one, which is a crude approximation to the time to last reference. Since the expected time to next reference is a monotonically increasing function of the time since last reference, the page to remove is one not referenced for some period τ in the past, which is precisely what working set does. Removal using LRU is somewhat more capricious than with working set, since whether a page is removed is a function of how many other pages have been referenced since it was last used. Working set thus uses the best possible estimator for the time to next reference among all algorithms with the given renewal property, and therefore is the best algorithm for the independent reference model among the class of such renewal algorithms. A_0 and Least Frequently Used, not belonging to this class of algorithms, are clearly superior.

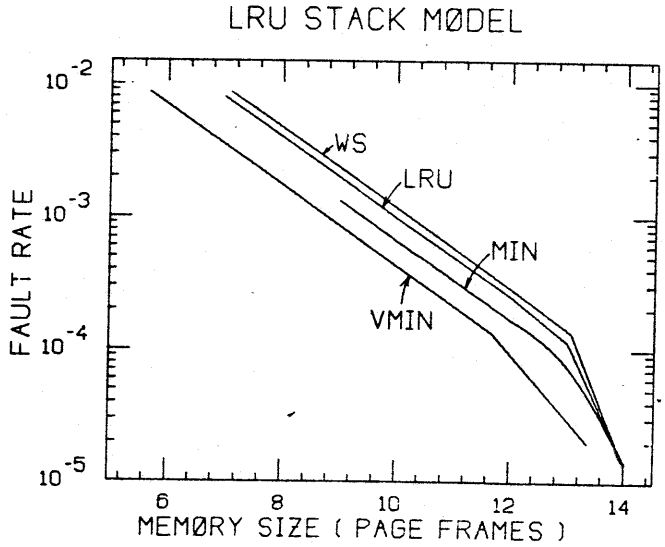

FIG. 4

PAGE FAULTS

We also o
held in core a
reader should
represent ratio

In Fig. 4 w
the working se
monotonically
algorithm (Spi
MIN as the opt
(by a factor of
amount.

For comp
simulated and
program writte
5) is taken from

The APL
addresses acce
simulation and
both the MIN
probabilities w
generated by th
in Fig. 5.

Because M
advantages ov
LRU for the L
This inherent a
our trace driver
approximately

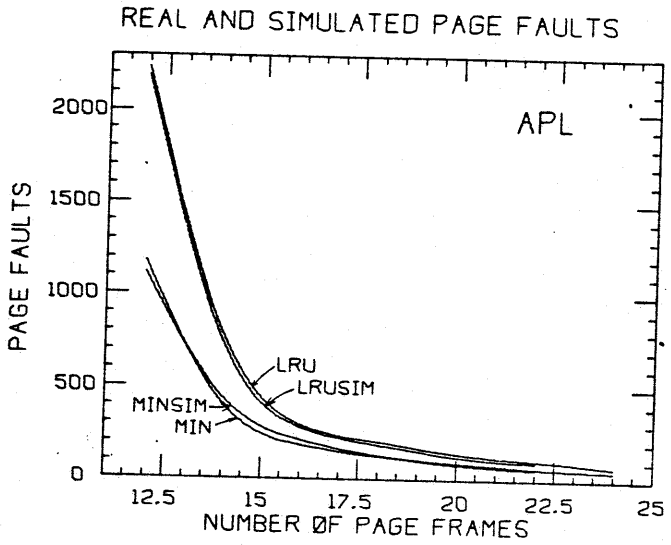

FIG. 5. (*From Baskett and Rafii* (1975))

We also observe, as expected, that VMIN, which varies the number of pages held in core according to the future behavior of the process, did the best. The reader should note that the vertical scale is logarithmic and thus vertical distances represent ratios.

In Fig. 4 we present the results of our simulation of the LRU stack model for the working set, LRU, MIN and VMIN algorithms. Our choice of $\{q_i\}$ convex and monotonically decreasing makes LRU the optimal realizable demand paging algorithm (Spirn (1973)), and, as expected, it performed better than working set. MIN as the optimal (unrealizable) algorithm can be seen to be significantly better (by a factor of about 35%), and VMIN is better than MIN by about the same amount.

For comparison with our simulation results, we show a comparison of simulated and measured results for the "APL" program, the execution of a program written in APL and described in detail in Smith (1974). This figure (Fig. 5) is taken from Baskett and Rafii (1975) and is reproduced with their permission.

The APL program was interpretively executed and a trace of the memory addresses accessed was produced. This trace was used to drive a trace driven simulation and the number of page faults generated by the trace was measured for both the MIN and the LRU algorithms. The observed set of LRU stack hit probabilities was then used to generate a reference string and the number of faults generated by the LRU and MIN algorithms was measured. The results are shown in Fig. 5.

Because MIN and VMIN are look-ahead algorithms, they have inherent advantages over optimal realizable algorithms, such as A_0 for the I.R.M. and LRU for the L.R.U.S.M. which can only minimize the fault rate on the average. This inherent advantage is apparent in our calculations, simulations and also in our trace driven simulation. Further, we note that the magnitude of this effect is approximately the same as the observed difference, in a number of studies,

between the results obtained with realizable and optimal algorithms. This is most clearly illustrated in Fig. 5 where the effect of MIN on a simulated reference string, generated from the LRU stack model, is almost identical to the effect on the real reference string.

6. Conclusions. We have demonstrated an algorithm for calculating the expected number of faults when a program is paged using either the MIN or VMIN replacement algorithm when that program can be described by either the LRU stack model or the independent reference model. This not only complements results by Denning and Schwartz (1972) and King (1971) on realizable algorithms, but it also demonstrates that it is possible to analyze look-ahead (nonrealizable) algorithms, certainly a nonobvious result. Although the complexity of our formulas makes direct calculation for most programs difficult or impossible, these exact results provide a basis from which a simple and effective approximation may be developed.

As noted in the last section, we have also shown that the difference in performance between realizable and nonrealizable algorithms on reference strings generated from our models of program behavior is very close to that seen on real program traces. This is all the more striking by being demonstrated analytically.

Acknowledgments. I'd like to thank Forest Baskett of Stanford University for pointing out the Denning, Savage and Spirn (1972) paper, and also to thank him and Abbas Rafii for permission to reproduce Fig. 5. Thanks are also due to Domenico Ferrari, Don Slutz and Ron Fagin for reading the manuscript and some helpful suggestions and to Ruth Suzuki for her excellent typing of this manuscript.

REFERENCES

A. V. AHO, P. J. DENNING AND J. D. ULLMAN (1971), *Principles of optimal page replacement*, J. Assoc. Comput. Mach., 18, p. 80–93.

F. BASKETT AND A. RAFII (1975), *Stochastic models of program paging behavior*, Rep., Stanford Univ. Computer Sci. Dept., Stanford, Calif., to appear.

L. A. BELADY (1966), *A study of replacement algorithms for a virtual storage computer*, IBM Systems J., 5, pp. 78–101.

L. A. BELADY AND F. P. PALERMO (1974), *On-line measurement of paging behavior by the multivalued MIN algorithm*, IBM J. Res. Develop., 18, pp. 2–19.

W. W. CHU AND H. OPDERBECK (1972), *The page fault frequency replacement algorithm*, Proceedings AFIPS 1972 Fall Joint Computer Conference, 46, no. 1, AFIPS Press, Montvale, N.J., pp. 597–609.

E. G. COFFMAN JR. AND P. J. DENNING (1973), *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, N.J.

D. R. COX (1962), *Renewal Theory*, Methuen, London (U.S. distrib. Barnes and Noble, New York).

P. J. DENNING (1968), *The working set model for program behavior*, Comm. ACM, 11, pp. 323–333.

———, (1972), *On modeling program behavior*, Proceedings AFIPS 1972, Spring Joint Computer Conference, AFIPS Press, Montvale, N.J., pp. 937–944.

P. J. DENNING, J. E. SAVAGE AND J. R. SPIRN (1972), *Models for locality in program behavior*, Tech. Rep. 107, Computer Sci. Lab., Dept. of Electrical Engrg., Princeton Univ., Princeton, N.J.

P. J. DENNING AND S. C. SCHWARTZ (1972), *Properties of the working-set model*, Comm. ACM, 15, pp. 191–198.

P. J. DENNING (1973), *Corrigendum to Denning and Schwartz* (1972), Comm. ACM, 16, p. 122.

P. A. FRANACZEK AND T. J. WAGNER (1974), *Some distribution free aspects of paging algorithm performance*, J. Assoc. Comput. Mach., 21, pp. 31–39.

T. KILBURN, D. B. G. EDWARDS, M. J. LANIGAN AND F. H. SUMNER (1962), *One-level storage system*, IRE Trans. Elec. Comp., EC-11, pp. 223–235.

W. F. KING III (1971), *Analysis of demand paging algorithms*, Proc. IFIPS Conf., Ljubljana, Yugoslavia, pp. TA-3-155–TA-3-160.

C. H. LEWIS AND R. A. NELSON (1974), *Some one pass algorithms for the generation of OPT distance strings*, IBM Rep. RC 4758.

P. A. W. LEWIS AND P. C. YUE (1971), *Statistical analysis of program reference patterns in a paging environment*, IEEE Comp. Soc. Conf., Boston, Mass., pp. 153–154.

P. A. W. LEWIS AND G. S. SHEDLER (1973), *Empirically derived micromodels for sequences of page exceptions*, IBM J. Res. Develop., pp. 86–100.

R. L. MATTSON, J. GECSEI, D. R. SLUTZ AND I. L. TRAIGER (1970), *Evaluation techniques for storage hierarchies*, IBM Systems J., 9, pp. 78–117.

P. H. ODEN AND G. S. SHEDLER (1972), *A model of memory contention in a paging machine*, Comm. ACM, 15, pp. 761–771.

J. E. POMERANZ (1971), *Paging with fewest expected replacements*, Proc. IFIPS Conf., Ljubljana, Yugoslavia, pp. TA-3-160–TA-3-162.

B. G. PRIEVE AND R. S. FABRY (1975), *An optimal variable space page replacement algorithm*, Proc. Fifth SIGOPS Conf., Austin, Tex., Nov. 1975; Comm. ACM, 19 (1976), pp. 295–297.

A. J. SMITH (1974), *A modified working set paging algorithm*, Rep. STAN-CS-74-451, Stanford Univ. Computer Sci. Dept., Stanford, Calif.; IEEE Trans. Computers, to appear Sept., 1976.

J. R. SPIRN (1973), *Program locality and dynamic memory management*, Ph.D. thesis, Dept. of Electrical Engrg., Princeton Univ., Princeton, N.J.