ON THE MODELING OF PAGING ALGORITHMS
BY FINITE AUTOMATA

C. C. Yang

Department of Information Sciences, University of Alabama in Birmingham
University Station Birmingham, Alabama 35294

(Received 9 January 1974)

Abstract—Two finite automata are devised for modeling two classes of demand paging algorithms. The first one of one input and three outputs models the class of algorithms each with a constant amount of allocated space. The second one of one input and six outputs models the class of algorithms each with a variable amount of allocated space. Some evaluation techniques are developed following each model. The memory states of the first class algorithm with the LRU, FIFO or FILO replacement policy or of the second class algorithm like the working set model or the page-turning rate algorithm are recursively defined by strings of the loaded pages. The adopted replacement policy and the state string updating procedure are imbedded in the recursive definition of a memory state. The operations like loading, tagging, reutilizing, etc. are reflected and differentiated by different outputs. Properties of some algorithms are developed. Some such properties confirm with previous results and some others are updated to fit the finiteness assumption of a reference string.

INDEX TERMS

Demand paging algorithm, finite automaton, inclusion property, memory management, page-turning rate algorithm, sequential machine, working set model, virtual storage computer system.

INTRODUCTION

In a virtual storage computer system, the memory allocation problem includes three aspects: (1) page fetching or loading, (2) page placement and (3) page replacement. The first one is the problem of deciding when a given page of a program in secondary memory is to be loaded into the allocated space of main storage. The second one is the problem of deciding where a loading page is to be placed. The last one is the problem of deciding which loaded page is to be selected and removed from the allocated space of main memory to make room for subsequent loading. In a paging environment, the placement strategy for placing k pages is just to use the replacement policy to free k allocated page frames. Placement is consequently irrelevant. In a demand paging system, loading is on demand only. Therefore, replacement is all that is left. In addition, besides the page size and the number of
allocated page frames, the replacement policy is one of the most important parameters in designing a virtual storage computer system. Aho, et al. discovered that minimum cost is always achieved by a demand paging algorithm under usual assumptions about memory system organization. Therefore, a study of demand paging algorithms with replacement policies deserves considerable interest and attention.

Although analytic studies of the algorithms just mentioned have yielded some important formal models like the working set model, the independent reference model, the stack algorithm, the probabilistic model based on a finite-state first-order Markov Chain, etc., the well established automata theory did not fruitfully serve in this area until the appearance of Gelenbe’s recent article in which he uses a stochastic automaton to formalize the class of the random partially preloaded algorithms.

This paper establishes two finite automata which can formalize two classes of demand paging algorithms: one involves a constant number of allocated page frames, like the Least-Recently Used algorithm and the other involves variable numbers of allocated page frames such as the working set model. Parameters for evaluating such algorithms and for investigating the properties of some such algorithms are developed. What is intended here is not to present results completely new to researchers in this area but rather to provide alternative mathematical models for better understanding the algorithms and for stimulating new ideas by an interdisciplinary approach. Besides this principal purpose, exploiting the applications area of abstract automata theory would be a by-product.

MODELING OF DEMAND PAGING ALGORITHMS BY FINITE AUTOMATA

A demand paging algorithm for the two-level memory management can be conveniently and effectively formalized by a mathematical model known as a sequential machine or transducer which is a finite automaton.

(1) Automata Model A

Let $P$ be the set of pages of a program and $K(P)$, the cardinality of $P$. The pages in $P$ are referenced one at a time so that the sequentially referenced pages form a sequence $(p_{t_1}, p_{t_2}, \ldots, p_{t_k}, \ldots)$ where each $p_{t_k}$ is a page in $P$. The concatenation of the sequence elements forms the reference string of the program which is denoted by

$$W_t=p_{t_1}p_{t_2}p_{t_3} \ldots p_{t_k} \ldots$$

(1)

The number of all concatenated pages in a reference string is called the length of the string. The length of $W_t$ is denoted as $L_t$ which is finite since $W_t$ is in the free monoid $P^*$ generated by $P$.

Suppose that $n$ page frames each accepting a page of equal size are allocated in main memory for processing a program. The set of loaded pages resided in the allocated space at reference instant of time $t$ constitutes the memory state $S_n(t)$ at $t$. By the usual set-theoretic approach, a memory state is represented by the set of loaded pages. However, in our proposed automata models, it is denoted by a string
of the loaded pages and the ordering of the string elements is determined by the adopted replacement policy of a demand paging algorithm. Consequently, two memory states are identical if and only if their corresponding strings are identical. At \( t=0 \), the allocated space is not loaded by any page in \( P \) so that it is referred to as the initial state \( S_n(0) \) and is denoted by \( \wedge \) which is a string of zero length. As the first referenced page \( P_{t_1} \) of \( W_r \) is loaded into one allocated frame, the initial state transits to its next state \( S_n(1) \) which is represented by the string \( P_{t_1} \) of length one and one \( q_i \) output is generated to indicate a single loading operation. The remaining pages of \( W_r \) are sequentially referenced, a sequence of states is produced by successive transitions and distinct referenced pages are loaded one at a time till the allocated space is saturated, i.e., the \( n \) allocated frames are all filled by \( n \) distinct referenced pages, or the \( K(P) \) distinct pages of a program are all loaded into a subset of the \( n \) allocated frames. After saturation, things become more complicated and will be discussed later. When either saturation condition just mentioned is satisfied, the memory state is called steady; otherwise, transient. The interval in which the initial state \( S_n(0) \) transits to the first steady state is known as the initial loading period for processing \( W_r \).

Let \( \text{Min}(n,K(P)) \) be the minimum between \( n \) and \( K(P) \). The \( n \) allocated frames can be loaded by at most \( \text{Min}(n,K(P)) \) distinct pages. A substring of \( W_r \) with length \( n \) can have at most \( \text{Min}(n,K(P)) \) distinct pages. Suppose \( L_t \) is the length of a substring of \( W_r \) corresponding to the referencing interval \( [t-L_t+1,t] \). Then there exists a map at the reference instant of time \( t \) which transforms the substring into the memory state \( S_n(t) \) such that

\[
L_t \geq \text{lg}(S_n(t)) \quad \text{for all } t
\]

(2a)

where \( \text{lg}(S_n(t)) \) denotes the length of the string for the state \( S_n(t) \). In addition the string of each steady state assumes a fixed length being equal to \( \text{Min}(n,K(P)) \), Thus we have

\[
L_t \geq \text{Min}(n,K(P))
\]

(2b)

for all steady states.

Let \( p_t \) in \( P \) be the current referenced page of \( W_r \) at reference instant of time \( t \). If \( p_t \) is in \( S_n(t-1) \), then \( S_n(t-1) \) transits to its next state \( S_n(t) \) and one \( q_i \) output which indicates that \( p_t \) is available for processing is emitted. The string of \( S_n(t) \) will not be identical with that of \( S_n(t-1) \) unless the ordering of \( p_t \) is not required to be updated replacement policy. On the other hand, if \( p_t \) is missing from \( S_n(t-1) \), then \( S_n(t-1) \) transits to the distinct state \( S_n(t) \) and one of the two possible outputs is emitted: \( q_i \) if \( S_n(t-1) \) is not saturated or \( q_o \) otherwise. Each output \( q_i \) indicates that some page in \( S_n(t-1) \) must be selected by the adopted replacement policy and removed to secondary storage to leave room in the allocated space for loading \( p_t \) into the freed frame. The adopted replacement policy also yields the string of \( S_n(t) \) by updating \( p_t \) and the string of \( S_n(t-1) \). Each output \( q_o \) emitted involves two activities: paging-out the selected page for replacement and loading or paging-in \( p_t \).
Later on, we may use the common symbol \( q_a \) to stand for either \( q_i \) or \( q_s \), since they are not necessarily to be differentiated in the sense of overheads produced. Besides this, a page-fault is also indicated by \( q_a \).

Based on the notions just described, it is readily seen that every demand paging algorithm with a constant number of allocated frames acts as a transformation device which transforms a reference string \( W_r \) into an equal-length output string \( W_o \) from which a loading string \( W_L \) can be produced. The string \( W_o \) is obtained from \( W_r \) by deleting those pages in \( W_r \) with corresponding outputs being \( q_o \). Intuitively, the effectiveness of this transformation device depends on the length \( L_r \) of the loading string \( W_L \); more specifically, the shorter the length \( L_r \), the better the device in the sense of running time required for accessing a program. Now, we formalize the algorithm mentioned above in the following:

A demand paging algorithm with constant allocated space and with a simple replacement policy is a sequential machine

\[ A=(R, P, Q_a, \Sigma_a, h_a, S_a(0)) \]

satisfying the following conditions:

(a) \( R \) is the set of memory states;

(b) \( P \) is the input alphabet;

(c) \( Q_a = \{ q_o, q_i, q_s \} \) is the output alphabet;

(d) The maps \( g_a \) and \( h_a \) are the next state function and the output function mapping from \( R \times P \) into \( R \) and onto \( Q_a \) respectively. State transition and output generation are determined by

\[
S_a(t) = g_a(S_a(t-1), p_t) \quad (3a)
\]

\[
q_a = h_a(S_a(t-1), p_a) \quad (3b)
\]

such that

(i) if \( p_t \) is in \( S_a(t-1) \), then \( p_t \) is also in \( S_a(t) \) and \( q_a = q_o \);

(ii) if \( p_t \) is not in \( S_a(t-1) \) and \( lg(S_a(t-1)) < lg(S_a(t)) \leq n \), then \( p_t \) is loaded into \( S(t) \) and \( q_a = q_i \);

(iii) if \( p_t \) is not in \( S_a(t-1) \) and \( lg(S_a(t-1)) = n \), then \( p_t \) replaces some loaded page and \( q_a = q_s \).

The string of \( S_a(t) \) is updated from \( p_t \) and that of \( S_a(t-1) \) and the updating procedure depends on the adopted replacement policy. Some loaded page for replacement under condition (d) (iii) is selected by the page-replacement rule and the tiedpage breaking rule if tied pages exist.

(e) \( S_a(0) = \lambda \) is the initial state.

The algorithms with the simple replacement policies like LRU (Least Recently Used), FIFO (First-In First-Out), FILO (First-In Last-Out), RAND (RANDom), etc., can be all formalized by this automata model.

During the initial loading period, only the \( q_i \) and \( q_s \) outputs can be generated as seen from conditions (d) (i) and (d) (ii) of the automata model \( A \). The total number \( \#q_i \) of \( q_i \) outputs for processing \( W_r \) would satisfy the following relation:
\[ #q_a = \min(n, K(P)) \]  \hfill (4)

The equation yields the following two consequences:

(a) When \( n = 1 \), every memory state except \( S_0(0) \) is steady and its string has only the corresponding referenced page. The loaded page to be selected for replacement is unique so that the adopted replacement policy is ineffective. In addition, the loading string \( W_n \) depends only on the program behavior which is characterized by \( W_f \).

(b) When \( n > K(P) \), the excessive allocated frames are wasted.

Based on both these facts, it is sufficient to consider only those cases in which

\[ 2 \leq n \leq K(P) \]  \hfill (5)

Since steady states are associated with the same number of allocated frames, it is significant to define the page-success rate \( s(n) \) and/or the missing-page rate \( m(n) \) for evaluating the relative merits of different algorithms under the same value of \( n \) and processing the same reference string \( W_f \) or for investigating the properties of an algorithm under different values of \( n \) or processing programs possessing different behaviors.

The page-success rate \( s(n) \) for processing \( W_f \) is defined to be the number of referenced pages not missing from the allocated space of main memory per reference instant of time. By the significance of the \( q_a \) output, we have

\[ s(n) = \frac{#q_a}{L_f} \]  \hfill (6a)

On the contrary, the page-fault rate \( f(n) \) for processing \( W_f \) can be defined as the number of page-faults per reference instant of time. Since a page-fault is reflected by the emission of one \( q_a \) output, this rate is

\[ f(n) = \frac{#q_a}{L_f} \]  \hfill (7a)

The missing-page rate \( m(n) \) for processing \( W_f \) which measures the number of referenced pages missing from the allocated space of main memory per reference instant of time is identical to \( f(n) \).

Since an automaton acts as a length-preserving transformation device, the sum of \#q_a and \#q_a must be equal to \( L_f \). Consequently, \( s(n) \) and \( f(n) \) can be alternatively expressed as

\[ s(n) = 1 - f(n) \]  \hfill (6b)

\[ f(n) = 1 - s(n) \]  \hfill (7b)

In addition, each output \( q_a \) reflects loading a page, the length \( L_a \) of the loading string is equal to the \#q_a of all \( q_a \) outputs. Thus we have

\[ f(n) = \frac{L_a}{L_f} \]  \hfill (7c)

To show that \( s(n) \) can have the other form besides (6a) and (6b), define some necessary definitions as follows: Suppose that a page \( p_t \) in \( P \) is referenced at both distinct instants of time \( t - x_t \) and \( t \). We say that the page \( p_t \) is recurrent at \( t \) (but not recurrent at \( t - x_t \)) with the interreference interval \( x_t \). Let \( L(t, n) \) be the longest sub-
string of \( W \), such that the number of distinct referenced pages over the interval \((t-L(t, n)+1, t)\) equals the length \( l_g(S_i(t)) \). Let \( N_r(n) \) be the number of all recurrent pages \( p_t \) (not necessarily distinct) at distinct reference instants of time \( t_j \) such that \( x_i \leq L(t, n) \) for all \( i \) and \( j \).

It should be noted that there may exist only one recurrent page \( p_t \) at each reference instant of time \( t_j \), although the number of interreference intervals of the page \( p_t \) over the interval \([t_j-L(t_j, n)+1, t_j]\) may not be unique. If the current referenced page \( p_t \) is recurrent at \( t \) with its interreference interval \( x_i \leq L(t, n) \), then \( L(t, n) = L(t-1, n)+1 \) and \( p_t \) was referenced at some instant of time \( t \) over the interval \((t-L(t, n)+1, t-1)\) or \((t-L(t-1, n), t-1)\) which corresponds to \( S_i(t-1) \), in other words, \( p_t \) is in \( S_i(t-1) \). From condition (d) (i) of the automaton A, the corresponding output must be \( q_0 \). This implies that

\[
N_r(n) = \#q_0 \tag{8}
\]

Hence, \( s(n) \) can be represented as

\[
s(n) = \frac{N_r(n)}{L_I} \tag{6c}
\]

Now, we establish the least upper bound of \( s(n) \) or the greatest lower bound of \( f(n) \). Firstly, the following statements are equivalent:

(a) \( n \geq K(P) \); \hfill (9a)

(b) \( \#q_2 = 0 \); \hfill (9b)

(c) \( \#q_1 = K(P) \). \hfill (9c)

If (9a) holds, then the first steady state has \( K(P) \) distinct pages of a program. Beyond the initial loading period, every page referenced at \( t \) is in the steady state \( S_n(t-1) \) so that condition (d) (iii) of the automaton A never occurs which in turn implies that (9b) holds. If \( \#q_2 = 0 \), then condition (d) (iii) of the automaton A never happens, in other words, no paging-out operation is taking place. This assures that each of the \( K(P) \) distinct pages of a program is loaded once and only once so that (9c) is established. That (9c) implies (9a) is obvious because of (4). Secondly, the upper bound of \( s(n) \) is

\[
s_{\text{max}} = 1 - \frac{K(P)}{L_I} \tag{10a}
\]

and the greatest lower bound of \( f(n) \) is

\[
f_{\text{min}} = \frac{K(P)}{L_I} \tag{11a}
\]

which hold if and only if (9a) holds. Both these bounds are established if and only if each of the \( K(P) \) pages of a program is loaded once and only once as described previously. Note that if \( n > K(P) \), the excessive allocated space is wasted so that the space is not efficiently utilized. If \( n = K(P) \), the replacement policy is ineffective since condition (d) (iii) of the automaton A never occurs. Thirdly, we examine the limiting values. From (10a) and (11a), if \( K(P) \) is relatively much smaller than \( L_I \), then

\[
s_{\text{max}} \rightarrow 1 \text{ if } K(P) \ll L_I \tag{10b}
\]
where the arrow denotes "approaches".

(2) Automata Model B

So far we have defined the automaton A which can model a class of demand paging algorithms with constant allocated spaces and with simple replacement policies. In the following, we model another class of demand paging algorithms each with variable allocated spaces. Let \( n(t) \) be the number of page frames allocated at reference instant of time \( t \). Then the value of \( n(t) \) should be determined by some means which can reflect the memory demand at \( t \), i.e.,

\[
n(t) = \log(S_P(t)) \quad \text{for all } t
\]  

where \( T \) is the window size associated with the interval \( [t-T+1, t] \). Based on the concept of memory demand, there exist three possible cases:

(a) \( n(t) > n(t-1) \);
(b) \( n(t) < n(t-1) \);
(c) \( n(t) = n(t-1) \).

Case (a) means that one more page frame is allocated at time \( t \).
Case (b) denotes that the allocated space dynamically contracts one page frame.
Case (c) implies that the allocated space is kept unchanged. For accommodating these cases, we need to define more operations besides the loading and paging-out activities. A tagging operation denoted by a \( q_i \) output means that a loaded page just leaves a memory state without actually paging-out. A reutilizing operation denoted by a \( q_i \) output means that a tagged page which is referenced returns to a memory state. Each output \( q_i' \) indicates that a loaded page is selected and tagged and the current referenced page is loaded. Each output \( q_i \) denotes that a loaded page is selected and tagged and the current referenced page is reused. Later on, a page being said to be missing from main memory means that the page resides in secondary storage. A page being said to be missing from a memory state implies that the page may be tagged in main memory or may still reside in secondary storage without ever paging-in. Now a second class algorithm can be formalized by the sequential machine

\[
B = (R, P, Q_b, g_b, h_b, S_P(0))
\]
satisfying the following conditions:

(a) \( R \) is the set of memory states;
(b) \( P \) is the input alphabet;
(c) \( Q_b = \{ q_0, q_1, q_2, q_3, q_4, q_5 \} \) is the output alphabet;
(d) The maps \( g_b \) and \( h_b \) are the next state function and the output function mapping from \( R \times P \) into \( R \) and onto \( Q_b \) respectively. State transition and output generation are determined by

\[
S_P(t) = g_b(S_P(t-1), p_t)
\]

\[
q_t = h_b(S_P(t-1), p_t)
\]
such that

(i) if $p_t$ is in $S_{p}(t-1)$ and $n(t)=n(t-1)$, then $p_t$ is also in $S_{p}(t)$ and $q_t=q_0$;
(ii) if $p_t$ is in $S_{p}(t-1)$ and $n(t)<n(t-1)$, then $p_t$ is also in $S_{p}(t)$ and some other page in $S_{p}(t-1)$ is tagged and $q_t=q_0$;
(iii) if $p_t$ is not in main memory and $n(t)>n(t-1)$, then $p_t$ is loaded into $S_{p}(t)$ and $q_t=q_0$;
(iv) if $p_t$ is not in main memory and $n(t)=n(t-1)$, then some page in $S_{p}(t-1)$ is tagged, $p_t$ is loaded into $S_{p}(t)$ and $q_t=q_0$;
(v) if $p_t$ is not in $S_{p}(t-1)$ but has been tagged and $n(t)>n(t-1)$, then $p_t$ returns to $S_{p}(t)$ and $q_t=q_0$;
(vi) if $p_t$ is not in $S_{p}(t-1)$ but has been tagged and $n(t)=n(t-1)$, then some page in $S_{p}(t-1)$ is tagged, $p_t$ returns to $S_{p}(t)$ and $q_t=q_0$.

(e) $S_{p}(0)=\emptyset$ is the initial state.

Based on the significances of the outputs in $Q_o$, the page-success rate $s(T)$, the page-fault rate $f(T)$ and the missing-page rate $m(T)$ can be represented in the following:

\[ s(T) = \frac{\#q_0 + \#q_1}{L_T} \quad (14a) \]
\[ f(T) = \frac{\#q_0}{L_T} \quad (15a) \]
\[ m(T) = \frac{\#q_0 + \#q_0}{L_T} \quad (16a) \]

where $q_0$ stands for $q_0$ or $q_1$ and $\#q_0$ is the number of all $q_0$ outputs. By the length-preserving property of an automaton, $s(T)$ and $m(T)$ can be alternatively denoted as

\[ s(T) = 1 - m(T) \quad (14b) \]
\[ m(T) = 1 - s(T) \quad (16b) \]

From (15a) and (16a), $m(T)$ is obviously an upper bond of $f(T)$. Due to the additional tagging and reutilizing operations introduced in the automata model B, each of the $K(P)$ distinct pages of a program is loaded once and only once, the number $\#q_0$ always assumes the minimum value $K(P)$ and accordingly $f(T)$ is always minimized and has the value $f_{\text{min}}$ as shown in (11a), i.e.,

\[ f(T) = f_{\text{min}} \quad (15b) \]

Since $m(T)$ is an upper bound of $f(T)$, we have

\[ m(T) \geq f_{\text{min}} \quad (16c) \]

which states that $m(T)$ is lower bounded by $f_{\text{min}}$. (10a), (11a), (14b) and (16c) imply that

\[ s(T) \leq s_{\text{max}} \quad (14c) \]

which states that $s(T)$ is also upper bounded by $s_{\text{max}}$. 

--- 56 ---
FORMALIZATION OF SOME DEMAND PAGING ALGORITHMS

So far we have defined two sequential machines which can model two classes of demand paging algorithms. Those models seem too general to specify which loaded page should be selected for subsequent replacement whenever a page-fault occurs or for tagging without actually paging-out and how the string of a memory state is updated, in other words, the adopted replacement policy for such an algorithm is not reflected in either model. However, this problem can be easily solved in this section by making some refinements. For a given algorithm with a specific replacement policy, the string of a memory state can be formed by a recursive definition. The page-replacement rule, the tied-page breaking rule if tied pages exist and the string updating procedure are implemented by the recursive definition. Firstly, we establish the recursive basis which is satisfied by every algorithm modeled by either automaton. At \( t = 1 \), when the referenced page is \( p_t \), the initial state represented by the null string \( \emptyset \) transits to its next state denoted by the string \( p_t \), and the output emitted is always \( q_t \), i.e.,

\[
(g(\emptyset, p_t), h(\emptyset, p_t)) = (p_t, q_t)
\]

(17)

where the map \( g \) stands for either \( g_a \) or \( g_s \) and \( h \), either \( h_a \) or \( h_b \). The \( g \) function in (17) constitutes the recursive basis for determining the strings of all memory states. Secondly, let \( w_t p_j \) for \( \langle w_0, p_j \rangle \) in \( P^* \times P \) be the string of the memory state at \( t-1 \) for \( t > 1 \). Denote the length of the substring \( w_t \) by \( \lg(w_t) \). Again, let \( p_t \) be the current referenced page in \( W_t \) at reference instant of time \( t \). Now, we are ready to perform fine formalization.

ALGORITHM WITH THE LRU REPLACEMENT POLICY

The LRU strategy replaces the loaded page which is the least recently referenced one by \( p_t \). The string of \( S_d(t) \) can be recursively determined by (17) and the \( g_a \) function operated on \( S_a(t-1) = w_t p_j \) and \( p_t \). The output generated is governed by the \( h_a \) function. Thus we have

\[
(g_a(w_t p_j, p_t), h_a(w_t p_j, p_t)) = \begin{cases} 
(p_t w_t, q_t), & \text{if } p_t = p_j, \\
(p_t (w_t - p_t) p_j, q_t), & \text{if } p_t \text{ is in } w_t, \\
(p_t w_t q_t, q_t), & \text{if } p_t \text{ is not in } w_t p_j \text{ and } \lg(w_t) < n - 1, \\
(p_t w_t q_t, q_t), & \text{if } p_t \text{ is not in } w_t p_j \text{ and } \lg(w_t) = n - 1 
\end{cases}
\]

(18a, 18b, 18c, 18d)

In (18b), \( w_t - p_t \) means that \( p_t \) is deleted from \( w_t \). The first element of each ordered-pair on the right-hand side is the string of \( S_d(t) \) with its leftmost element being always the most recently referenced page \( p_t \) and its rightmost element being the least recently referenced page.

ALGORITHM WITH THE FIFO REPLACEMENT POLICY

As the name FIFO implies, this strategy replaces the least recently loaded page by \( p_t \). State transition and output emission are determined by (17) and the following:
\[(g_d(w_i p_j, q_i), h_d(w_i p_j, p_i)) = \begin{cases} (w_i p_j, q_i), & \text{if } p_i \text{ is in } w_i p_j \\ \text{right-hand side of (18c)} & \\
\text{right-hand side of (18d)} & \end{cases} \] (19a) (19b) (19c)

As seen from (18) and (19), the only difference between the LRU and FIFO policies exists when the output generated is \( q_i \). We may consider that the LRU algorithm is a refinement of the FIFO algorithm since the string updating procedure is refined whenever \( q_i \) is emitted.

**THE ALGORITHM WITH THE FILO OR LIFO REPLACEMENT POLICY**

In contrast to the FIFO algorithm, this policy replaces the most recently loaded page by \( p_i \). State transition and output emission can be determined by (17) and the following:

\[(g_d(w_i p_j, p_i), h_d(w_i p_j, p_i)) = \begin{cases} (w_i p_j, q_i), & \text{if } p_i \text{ is not in } w_i p_j \text{ and } lg(w_i) < n - 1 \\
(w_i p_j, q_i), & \text{if } p_i \text{ is not in } w_i p_j \text{ and } lg(w_i) = n - 1 \end{cases} \] (20a) (20b) (20c)

This algorithm favors or biases all pages in \( w_i \) where the head of \( w_i \) is the first loaded page or the head reference string \( W_i \). However, the sense of biasing here is not the same with that of the biased replacement algorithm proposed by Belady.

**THE WORKING SET MODEL**

This model was proposed by Denning based on the principle of locality. The model provides an adaptive estimator for memory demand. At each reference instant of time, the substring of \( W_i \) with length \( T \) over the interval \([t-T+1, t]\) maps into the memory state \( S_p(t) \) such that (12) is satisfied. The set of distinct pages in the substring is the working set \( W(t, T) \) at \( t \). Thus the following statements are obviously equivalent:

(a) the working set size \( w(t, T) \) which is the cardinality of \( W(t, T) \);
(b) the length of the string of \( S_p(t) \), i.e., \( lg(S_p(t)) \);
(c) \( m(t) \), the number of allocated frames at \( t \).

The page-replacement rule and the string updating procedure adopted in the working set model are the same as those in the LRU algorithm. Hence, the working set model is nothing but a demand paging algorithm with the LRU replacement policy and with variable numbers of allocated frames. It maps substrings of \( W_i \) with a constant length \( T \) into steady memory states of variable state string lengths. However, the LRU algorithm described previously maps substrings of \( W_i \) with variable lengths into steady memory states of a constant state string length. State transition and output generation can be determined by (17) and the following:
\[(g_0(w, p, p_t), h_0(w, p, p_t))\]

right-hand side of (18a) \hfill (21a)

right-hand side of (18b), if \(w(t, T) = w(t-1, T)\) \hfill (21b)

\((p_t, w, p, q_i)\), if \(p_t\) is not in main memory and \(w(t, T) > w(t-1, T)\) \hfill (21c)

\[
\begin{cases}
(p_t, w, q_i), & \text{if } p_t \text{ is not in main memory and } w(t, T) = w(t-1, T) \\
(p_t, w - p_t, q_i), & \text{if } p_t \text{ is in } w_t \text{ and } w(t, T) > w(t-1, T) \\
(p_t, w_i, p, q_i), & \text{if } p_t \text{ is not in } W(t-1, T) \text{ but has been tagged and } w(t, T) > w(t-1, T) \\
(p_t, w, q_i), & \text{if } p_t \text{ is not in } W(t-1, T) \text{ but has been tagged and } w(t, T) = w(t-1, T)
\end{cases}
\hfill (21d)-(21g)

THE PAGE-TURNING RATE ALGORITHM

Besides the working set model, this algorithm can also reveal memory demand at each reference instant of time \(t\) by controlling the page-turning rate within a certain limit. Firstly, we define the page-turning rate. Let \(q_d(t)\) and \(q_d(t, T)\) be the numbers of the \(q_i\) and \(q_i'\) outputs generated over the intervals \([1, t]\) for \(1 \leq t \leq T\) and \([t-T+1, t]\) for \(1 \leq T \leq t\) respectively. The page-turning rate \(\alpha(t, T)\) at time \(t\) is defined as

\[
\alpha(t, T) = \begin{cases}
\frac{q_d(t)}{t}, & \text{if } 1 \leq t \leq T \\
\frac{q_d(t, T)}{T}, & \text{if } 1 \leq T \leq t
\end{cases}
\hfill (22a)-(22b)

At \(t=0\), the page-turning rate is defined as 0, i.e.,

\[
\alpha(0, T) = 0 \text{ for all } T
\hfill (22c)
\]

When \(T=0\), the page-turning rate is also defined as 0, i.e.,

\[
\alpha(t, 0) = 0 \text{ for all } t
\hfill (22d)
\]

By this algorithm, a criterion

\[
\frac{b}{T} \geq \alpha(t, T) \geq \frac{a}{T}, \ 0 < a, b < T
\hfill (23)
\]

is adopted for controlling the page-turning rate from time to time. By this method, the memory demand at each reference instant of time \(t\) can be determined as follows:

(a) When \(p_t\) is in \(S_p(t-1)\),

\[
n(t) = \begin{cases}
n(t-1), & \text{if } \alpha(t-1, T) \geq \frac{a}{T} \\
n(t-1) - 1, & \text{if } \alpha(t-1, T) < \frac{a}{T}
\end{cases}
\hfill (24a)-(24b)
\]

(b) When \(p_t\) is not in \(S_p(t-1)\),

\[
n(t) = \begin{cases}
n(t-1) + 1, & \text{if } \alpha(t-1, T) \geq \frac{b}{T} \\
n(t-1), & \text{if } \alpha(t-1, T) < \frac{b}{T}
\end{cases}
\hfill (24c)-(24d)
\]

--- 59 ---
By (12) and (24), this algorithm can be easily modeled by the automaton B. State transition and output generation can be similarly determined as (21) if the LRU replacement policy is adopted.

The page-fault rate, the missing-page rate and the page-success rate depend not only on $T$ but also on $a$ and $b$. How to choose appropriate values for these three parameters for achieving a better performance would be the key point to be considered. From (15b) and (11a), the following relation should be maintained:

$$1 > \frac{b}{T} \rightarrow \frac{a}{T} \geq \frac{K(P)}{L_f} \quad (25)$$

However, the value of the lower bound in (25) is not available unless the reference string is generated. Thus the determination of the values of these parameters is subject to experimentation. It should be noted that a special case of this algorithm can be yielded if $a=b$ and this case can be modeled by the automaton A.

**PROPERTIES OF SOME DEMAND PAGING ALGORITHMS**

**The LRU Algorithm**

Table 1 illustrates the LRU algorithm for various values of $n$. The reference string $W_t$ is identical to that in [5]. The string for each memory state is represented by a list whose top element is $p_r$. Each such list is a LRU stack as defined by Mattson, et al. The page success rate $s(n)$ which is equivalent to the success function in [5] is directly determined by (6a) without utilizing the stack distances. The string

<table>
<thead>
<tr>
<th>Table 1. Illustration of the LRU Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
or stack updating procedure is imbedded in the recursive definition for the string of a memory state.

As shown by Mattson et al\textsuperscript{9}, the LRU strategy belongs to the class of stack algorithms each of which has an important property known as the inclusion property between a pair of states $S_n(t)$ and $S_{n+k}(t)$ for $k \geq 1$ such that every page in $S_{n+k}(t)$ is also in $S_n(t)$. This property exists for the LRU algorithm and can be easily verified by means of the automata model $A$ as shown in Appendix A. This property in turn assures that the page-success rate $s(n)$ is nondecreasing with $n$. To show this latter property, firstly, the inclusion property between $S_{n+k}(t)$ and $S_n(t)$ implies that

$$l_g(S_{n+k}(t)) \geq l_g(S_n(t)) \text{ for all } t$$

(26) Secondly, by the definition of $L(t,n)$, we have

$$L(t,n+1) \geq L(t,n) \text{ for all } t$$

(27) Thirdly, by the definition of $N_r(n)$, we have

$$N_r(n+1) \geq N_r(n)$$

(28) Finally, (8), (6c), (10a) and (28) yield the following conclusion:

$$s_{\text{max}} \geq s(n+1) \geq s(n) \geq s(1) \geq 0$$

(29) The FIFO Algorithm

Belady et al\textsuperscript{10} first discovered that there exists an anomaly in space-time characteristics of certain reference strings processed by the FIFO algorithm, i.e., a decrease in $n$ is accompanied by an increase in $s(n)$. This anomaly occurs if there exists some pair of states having no inclusion property. The verification by means of the automata model $A$ is shown in Appendix B.

The Working Set Model

Denning et al\textsuperscript{11} discovered nine important properties of the working set model by means of a probabilistic approach under some assumptions. Eight of them can be developed by the automata model $B$ and updated to fit any reference string of finite length.

Before displaying those properties, we define some definitions. Similar to $N_r(n)$ and $s(n)$, the page-success rate $s(T)$ of the working set model can be represented as

$$s(T) = \frac{N_r(T)}{L}$$

(14d) where $N_r(T)$ is the number of all recurrent pages (not necessarily distinct) $p_t$ with $1 \leq x_t \leq T$ for all $p_t$ referenced at distinct instants of time. Define the missing-page indicator $A$ which assumes binary values as

$$A(t-1,T) = \begin{cases} 1, & \text{if } h_b(S_r(t-1), p_t) = q_a \text{ or } q_b \\ 0, & \text{otherwise} \end{cases}$$

(30a) At $t=1$, the output emitted is always $q_i$ as shown in (17) so that (30a) holds and yields

$$A(0,T) = 1 \text{ for all } T$$

(30c)
At \( t=L_{r}+1 \), no page is referenced or the referenced page is the identity \( \land \) of the free monoid \( P^* \). Thus no output is generated or the emitted output is the identity \( \land \), of the free monoid \( Q_{s}^* \) generated by \( Q_s \) since \( h_s(S_{r}(t), \land) = \land \) for any \( t \) holds if the output function \( h_s \) is extended such that it maps from \( R \times P^* \) into \( Q_{s}^* \). (30b) implies that

\[
A(L_{r}, T) = 0 \text{ for all } T \quad (30d)
\]

By means of this indicator, it is not hard to establish the following form of the missing-page rate as

\[
m(T) = \frac{1}{L_{r}} \sum_{i=1}^{L_{r}} A(t-1, T) \quad (16d)
\]

Since the states \( S_{r}(t-1) \) and \( S_{r+1}(t) \) are respectively mapped from the substrings \( \rho_{L} \rho_{r+1} \ldots \rho_{t-1} \) and \( \rho_{1} \rho_{r} \rho_{r+1} \ldots \rho_{t} \), the following relation which is similar to (3.1) in [11] is established by (30a) and (30b), i.e.,

\[
w(t, T+1) = w(t-1, T) + A(t-1, T) \quad (31)
\]

Define the average working set size \( w(T) \) for processing the reference string \( W_{r} \) of a finite length \( L_{r} \) as

\[
w(T) = \frac{1}{L_{r}} \sum_{i=1}^{L_{r}} w(t, T) \quad (32a)
\]

where \( t \) may start from either 0 or 1 since \( w(0, T) = 0 \) for all \( T \). The working set size \( w(t, 0) = 0 \) for all \( t \) implies that

\[
w(0) = 0 \quad (32b)
\]

Then the properties developed by the automata approach entail the following:

\[
P1: \quad 1 = w(0) \leq w(T) \leq w(T+1) \leq \min(T+1, K(P)) \quad (33)
\]

\[
P2: \quad w(T+1) - w(T) = m(T) - \frac{1}{L_{r}} w(L_{r}, T) \quad (34)
\]

\[
P3: \quad 0 < f_{\text{min}} \leq m(T+1) \leq m(T) \leq m(0) = 1 \quad (35)
\]

\[
P4: \quad m(T) = 1 - s(T) \quad (16b)
\]

\[
P5: \quad m(T+1) - m(T) = -s'(T+1) \quad (36)
\]

where \( s'(T+1) = s(T+1) - s(T) \quad (37)\)

\[
P6: \quad w(T) = \sum_{\bar{z}=1}^{L_{r}} m(\bar{z}) - \frac{1}{L_{r}} \sum_{\bar{z}=0}^{L_{r}-1} w(L_{r}, \bar{z}) \quad (38)
\]

\[
P7: \quad w(T+1) + w(T-1) \leq 2w(T) - \frac{1}{L_{r}} [w(L_{r}, T) - w(L_{r}, T-1)] \quad (39)
\]

\[
P9: \quad \lim_{T \to \infty} m(T) = f_{\text{min}} \quad (40)
\]

All these properties just mentioned can be easily verified by the similar arguments taken in [11] without utilizing the concept of probability. Some of them are demonstrated in Table 2. The other property \( P8 \) shown below
Table 2. Illustrating the Properties of the Working Set Model

<table>
<thead>
<tr>
<th>$T$</th>
<th>$t$</th>
<th>$W_T$</th>
<th>$f(T)$</th>
<th>$s(T)$</th>
<th>$m(T)$</th>
<th>$\sum_{x=2}^{T-1} m(x)$</th>
<th>$w(T)$</th>
<th>$N_r(T)$ pages $p_i$ recurrent at $t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>$a b b c b a d c a a$</td>
<td>.40</td>
<td>.20</td>
<td>.80</td>
<td>1.00</td>
<td>1.00</td>
<td>$(p_1,p_2)$</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>$a b c d$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>$a b c d$</td>
<td>.40</td>
<td>.30</td>
<td>.70</td>
<td>1.80</td>
<td>1.70</td>
<td>$(p_1,p_2)$</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>$a b c d$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>$a b c d$</td>
<td>.40</td>
<td>.40</td>
<td>.60</td>
<td>2.50</td>
<td>2.30</td>
<td>$(p_1,p_2,p_3)$</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>$a b c d$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>$a b c d$</td>
<td>.40</td>
<td>.60</td>
<td>.60</td>
<td>3.60</td>
<td>2.90</td>
<td>$(p_1,p_2,p_3)$</td>
</tr>
</tbody>
</table>

---

March, 1974

Yang: On the Modeling of Paging Algorithms by Finite Automata
$P8: \lim_{T \to \infty} \omega(T) = N_r$ (41)

where $N_r$ is the number of distinct recurrent pages in a reference string $W_r$ as yet not verified here.

DISCUSSIONS

This paper has presented two finite automata models: one can formalize a class of demand paging algorithms each with a constant allocated space and the other for a class of algorithms each with variable allocated spaces. The LRU, FIFO and FILO algorithms of the first class are formulated in detail by recursive definitions for their memory states represented by strings and so are the working-set model and the page-turning rate strategy of the second class. The state string updating procedure, and the page-replacement rule of the adopted replacement policy are imbedded in the recursive definition. The page-success rate of the LRU strategy can be directly determined by the number of $q_0$ outputs emitted and the length of reference string processed without utilizing the concept of stack distance. The LRU algorithm which has the inclusion property and the FIFO strategy which may not have this property are verified in Appendices A and B. The page-turning rate algorithm which also has the capability of revealing memand demand is introduced. Properties of the working set model for finite reference strings are developed. Some of them are confirmed to previous results for the case of infinite reference strings. The remaining properties of the previous results are updated to fit the finiteness assumption. Since automata theory is very well established, we might be able to borrow more concepts from this area to stimulate new ideas and techniques for solving problems related to resource management of computer systems under multiprogramming, multiprocessing and/or time-sharing environments.

ACKNOWLEDGEMENT

This work was started in IBM Research Laboratory, San Jose, California while the author was visiting there one year in 1971-72. The author wishes to thank Mrs. Barbara Brawn (now Mrs. Barbara Matton) for her encouragement throughout the author’s year in IBM San Jose Research Laboratory.

APPENDIX A

Verification of the Inclusion Property for the LRU Algorithm

Consider the following four possibilities:

(1) $S_n(t-1)$ and $S_{n+1}(t-1)$ are both transient.

Under this condition, $S_n(t-1) = S_{n+1}(t-1)$ since (18d) has never been applied. Suppose that $S_n(t-1) = w_m$ be the last transient state and $p_t$, the current referenced page not in $w_m$. Then $S_n(t) = p_t w_m$ is the first steady state by (18d) and $S_{n+1}(t) = p_t w_m$ is also a transient state by (18c).

(2) $S_n(t-1)$ is steady and $S_{n+1}(t-1)$ is transient.
Under this condition, \( S_n(t-1) = S_{n+1}(t-1) \) as seen above. Suppose that \( S_{n+1}(t-1) = w_1 p_j \) be the last transient state and \( p_t \) be the current referenced page not in \( w_1 p_j \). Then \( S_{n+1}(t) = p_t w_1 p_j \) by (18c) and \( S_n(t) = p_t w_2 \) by (18d).

(3) \( S_n(t-1) \) and \( S_{n+1}(t-1) \) are both steady. Suppose that \( S_n(t-1) = w_2 \) and \( S_{n+1}(t-1) = w_n p_j \). Then

\[
S_n(t) = \begin{cases} 
  p_t (w_n - p_t), & \text{if } p_t \text{ is in } w_n \\
  p_t (w_n - p_1), & \text{if } p_t \text{ is not in } w_n
\end{cases} \tag{A1a}
\]

and

\[
S_{n+1}(t) = \begin{cases} 
  p_t (w_n p_j - p_t), & \text{if } p_t \text{ is in } w_n p_j \\
  p_t w_n, & \text{if } p_t \text{ is not in } w_n
\end{cases} \tag{A2b}
\]

(4) \( S_n(t-1) \) is transient and \( S_{n+1}(t-1) \) is steady. This is an impossible case.

**APPENDIX B**

The FIFO Algorithm May Not Have the Inclusion Property

The following assumptions are firstly made:

(1) Let \( a \) and \( b \) be the numbers of allocated frames such that \( 1 < a < b \).

(2) Let \( S_n(t-1) = w_m p_t \) and \( S_{n+1}(t-1) = w_n p_j \) be the two states at \( t-1 \) such that

(a) both states are steady,

(b) \( p_j \) be in \( w_m \)

(c) each element in \( w_m p_t \) be also in \( w_n p_j \).

(3) \( p_t \) be the current referenced page not in \( w_n p_j \).

The states \( S_n(t-1) \) and \( S_{n+1}(t-1) \) satisfy the inclusion property because of assumption (2) (c). The state \( S_n(t-1) \) being assumed to be steady in assumption (2) (a) and assumption (3) imply that (19c) or (18d) holds which in turn yields

\[
S_n(t) = p_t w_n \tag{B1}
\]

Assumptions (2) (c) and (3) imply that \( p_t \) is not in \( w_m p_t \). Since \( S_n(t-1) \) is also steady, we have

\[
S_n(t) = p_t w_m \tag{B2}
\]

By assumption (2) (b) and (B2), the page \( p_j \) which is in \( S_n(t) \) does not appear in \( w_n \) (since \( S_n(t-1) = w_n p_j \) or in \( S_n(t) \) as shown in (B1)). Thus the inclusion property is not satisfied for the next states at time \( t \).

**FOOTNOTES**

This work was supported in part by IBM Taiwan Corporation, IBM World Trade Corporation, and IBM San Jose Research Laboratory and by N.I.H. Grant No. 5-501-RR05300-12 at University of Alabama in Birmingham.

The author is with the Department of Information Sciences, University of Alabama in Birmingham, University Station, Birmingham, Alabama 35294.
REFERENCES