Science Bulletin of NCTU, Vol. VII, No. 1

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[1], the memory allocation problem includes three aspects[2]: (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[1] in designing a virtual storage computer system. Aho, *et al*[3] 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[4], the independent reference model[3], the stack algorithm[5], the probabilistic model based on a finite-state first-order Markov Chain[6], etc., the well established automata theory[7] did not fruitfully serve in this area until the appearance of Gelenbe's recent article[8] 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*[7].

(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 $(p_{t_1}, p_{t_2}, \ldots\ldots, p_{t_k}, \ldots\ldots)$ 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

$$W_I = p_{t_1} p_{t_2} p_{t_3} \ldots\ldots p_{t_k} \ldots\ldots \qquad (1)$$

The number of all concatenated pages in a reference string is called the *length* of the string. The length of W_I is denoted as L_I which is finite since W_I 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* $S_n(t)$ and is denoted by $\wedge$ which is a string of zero length. As the first referenced page p_{i_1} of W_I is loaded into one allocated frame, the initial state transits to its next state $S_n(1)$ which is represented by the string p_{i_1} of length one and one q_1 output is generated to indicate a single *loading operation*. The remaining pages of W_I 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_I.

Let $Min(n, K(P))$ be the minimum between n and $K(P)$. The n allocated frames can be loaded by at most $Min(n, K(P))$ distinct pages. A substring of W_I with length n can have at most $Min(n, K(P))$ distinct pages. Suppose L_t is the length of a substring of W_I 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

$$L_t \geq lg(S_n(t)) \quad \text{for all } t \tag{2a}$$

where $lg(S_n(t))$ 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 $Min(n, K(P))$, Thus we have

$$L_t \geq Min(n, K(P)) \tag{2b}$$

for all steady states,

Let p_t in P be the *current referenced page* of W_I 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_0 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_1 if $S_n(t-1)$ is not saturated or q_2 otherwise. Each output q_2 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_2 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_1 or q_2 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_I into an equal-length *output string* W_0 from which a *loading string* W_L can be produced. The string W_L is obtained from W_I by deleting those pages in W_I with corresponding outputs being q_0. Intuitively, the effectiveness of this transformation device depends on the length L_L of the loading string W_L; more specifically, the shorter the length L_L, the better the device in the sense of running time required for paocessing 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

$$A=(R, P, Q_a, g_a, h_a, S_n(0))$$

satisfying the following conditions:

(a) R is the set of memory states;

(b) P is the *input alphabet*;

(c) $Q_a=\{q_0, q_1, q_2\}$ 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

$$S_n(t)=g_a(S_n(t-1), p_t) \tag{3a}$$

$$q_t=h_a(S_n(t-1), p_t) \tag{3b}$$

such that

(i) if p_t is in $S_n(t-1)$, then p_t is also in $S_n(t)$ and $q_t=q_0$;

(ii) if p_t is not in $S_n(t-1)$ and $lg(S_n(t-1))<lg(S_n(t))\leq n$, then p_t is loaded into $S(t)$ and $q_t=q_1$;

(iii) if p_t is not in $S_n(t-1)$ and $lg(S_n(t-1))=n$, then p_t replaces some loaded page and $q_t=q_2$.

The string of $S_n(t)$ is updated from p_t and that of $S_n(t-1)$ 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) $S_n(0)=\wedge$ 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_1 and q_0 outputs can be generated as seen from conditions (d) (i) and (d) (ii) of the automata model A. The total number $\#q_1$ of q_1 outputs for processing W_I would satisfy the following relation:

$$\#q_1 = Min(n, K(P)) \tag{4}$$

The equation yields the following two consequences:

(a) When $n=1$, every memory state except $S_n(0)$ 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_L depends only on the program behavior which is characterized by W_I.

(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

$$2 \leq n \leq K(P) \tag{5}$$

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_I 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_I 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_0 output, we have

$$s(n) = \frac{\#q_0}{L_I} \tag{6a}$$

On the contrary, the *page-fault rate* $f(n)$ for processing W_I 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

$$f(n) = \frac{\#q_a}{L_I} \tag{7a}$$

The *missing-page rate* $m(n)$ for processing W_I 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 $\#q_0$ and $\#q_a$ must be equal to L_I. Consequently, $s(n)$ and $f(n)$ can be alternatively expressed as

$$s(n) = 1 - f(n) \tag{6b}$$

$$f(n) = 1 - s(n) \tag{7b}$$

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

$$f(n) = \frac{L_L}{L_I} \tag{7c}$$

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_i in P is referenced at both distinct instants of time $t-x_i$ and t. We say that the page p_i is *recurrent* at t (but not recurrent at $t-x_i$) with the *interreference interval* x_i. Let $L(t, n)$ be the longest sub-

string of W_I such that the number of distinct referenced pages over the interval $[t-L(t,n)+1, t]$ equals the length $lg(S_n(t))$. Let $N_r(n)$ be the number of all recurrent pages p_i (not necessarily distinct) at distinct reference instants of time t_j such that $x_i \leq L(t_j, n)$ for all i and j.

It should be noted that there may exist only one recurrent page p_i at each reference instant of time t_j although the number of interreference intervals of the page p_i 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 $x_t \leq L(t,n)$, then $L(t,n)=L(t-1,n)+1$ and p_t was referenced at some instant of time over the interval $[t-L(t,n)+1, t-1]$ or $[t-L(t-1,n), t-1]$ which corresponds to $S_n(t-1)$, in other words, p_t is in $S_n(t-1)$. 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, $s(n_0)$ 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) $n \geq K(P)$; (9a)

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

(c) $\#q_1=K(P)$. (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

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

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

$$f_{min}=\frac{K(P)}{L_I} \tag{11a}$$

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

$$s_{max} \to 1 \text{ if } K(P) \ll L_I \tag{10b}$$

$$f_{min} \to 0 \text{ if } K(P) \ll L_I \tag{11b}$$

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.*,

$$n(t)=lg(S_T(t)) \text{ for all } t \tag{12}$$

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 accomodating these cases, we need to define more operations besides the loading and paging-out activities. A *tagging operation* denoted by a q_3 output means that a loaded page just leaves a memory state without actually paging-out. A *reutilizing operation* denoted by a q_4 output means that a tagged page which is referenced returns to a memory state. Each output q_2' indicates that a loaded page is selected and tagged and the current referenced page is loaded. Each output q_5 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

$$B=(R, P, Q_b, g_b, h_b, S_T(0))$$

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

$$S_T(t)=g_b(S_T(t-1)p_t) \tag{13a}$$

$$q_t=h_b(S_T(t-1), p_t) \tag{13b}$$

such that

(i) if p_t is in $S_T(t-1)$ and $n(t)=n(t-1)$, then p_t is also in $S_T(t)$ and $q_t=q_0$;

(ii) if p_t is in $S_T(t-1)$ and $n(t)<n(t-1)$, then p_t is also in $S_T(t)$ and some other page in $S_T(t-1)$ is tagged and $q_t=q_3$;

(iii) if p_t is not in main memory and $n(t)>n(t-1)$, then p_t is loaded into $S_T(t)$ and $q_t=q_1$;

(iv) if p_t is not in main memory and $n(t)=n(t-1)$, then some page in $S_T(t-1)$ is tagged, p_t is loaded into $S_T(t)$ and $q_t=q_2'$;

(v) if p_t is not in $S_T(t-1)$ but has been tagged and $n(t)>n(t-1)$, then p_t returns to $S_T(t)$ and $q_t=q_4$;

(vi) if p_t is not in $S_T(t-1)$ but has been tagged and $n(t)=n(t-1)$, then some page in $S_T(t-1)$ is tagged, p_t returns to $S_T(t)$ and $q_t=q_5$.

(e) $S_T(0)=\Lambda$ is the initial state.

Based on the significances of the outputs in Q_b, 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:

$$s(T)=\frac{\#q_0+\#q_3}{L_T} \tag{14a}$$

$$f(T)=\frac{\#q_a}{L_T} \tag{15a}$$

$$m(T)=\frac{\#q_a+\#q_b}{L_T} \tag{16a}$$

where q_b stands for q_4 or q_5 and $\#q_b$ is the number of all q_b outputs. By the length-preserving property of an automaton, $s(T)$ and $m(T)$ can be alternatively denoted as

$$s(T)=1-m(T) \tag{14b}$$

$$m(T)=1-s(T) \tag{16b}$$

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 $\#q_a$ always assumes the minimum value $K(P)$ and accordingly $f(T)$ is always minimized and has the value f_{min} as shown in (11a), *i. e.*,

$$f(T)=f_{min} \tag{15b}$$

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

$$m(T)\geq f_{min} \tag{16c}$$

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

$$s(T)\leq s_{max} \tag{14c}$$

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

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 $\wedge$ transits to its next state denoted by the string p_t, and the output emitted is always q_1, *i.e.*,

$$(g(\wedge, p_t), h(\wedge, p_t))=(p_t, q_1) \tag{17}$$

where the map g stands for either g_a or g_b 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_tp_j for (w_t, p_j) 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_I 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 $S_n(t)$ can be recursively determined by (17) and the g_a function operated on $S_n(t-1)=w_tp_j$ and p_t. The output generated is governed by the h_a function. Thus we have

$$(g_a(w_tp_j, p_t), h_a(w_tp_j, p_t))=\begin{cases}(p_tw_t, q_0), \text{ if } p_t=p_j, & \text{(18a)}\\ (p_t(w_t-p_t)p_j, q_0), \text{ if } p_t \text{ is in } w_t, & \text{(18b)}\\ (p_tw_tp_j, q_1) \text{ if } p_t \text{ is not in } w_tp_j \text{ and } lg(w_t)<n-1, & \text{(18c)}\\ (p_tw_t, q_2), \text{ if } p_t \text{ is not in } w_tp_j \text{ and } lg(w_t)=n-1 & \text{(18d)}\end{cases}$$

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 $S_n(t)$ 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:

$$(g_a(w_ip_j, p_t), h_a(w_ip_j, p_t)) = \begin{cases} (w_ip_j, q_0), \text{ if } p_t \text{ is in } w_ip_j & (19a) \\ \text{right-hand side of (18c)} & (19b) \\ \text{right-hand side of (18d)} & (19c) \end{cases}$$

As seen from (18) and (19), the only difference between the LRU and FIFO policies exists when the output generated is q_0. We may consider that the LRU algorithm is a refinement of the FIFO algorithm since the string updating procedure is refined whenever q_0 is emitted.

THE ALGORITHM WITH THE FILO OR LIFO RFPLACEMENT POLICY

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

$$(g_a(w_ip_j, p_t), h_a(w_ip_j, p_t)) = \begin{cases} \text{right-hand side of (19a)} & (20a) \\ (w_ip_jp_t, q_1), \text{ if } p_t \text{ is not in } w_ip_j \text{ and } lg(w_i) < n-1 & (20b) \\ (w_ip_j, q_2), \text{ if } p_t \text{ is not in } w_ip_j \text{ and } lg(w_i) = n-1 & (20c) \end{cases}$$

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 replacemement algorithm proposed by Belady[9].

THE WORKING SET MODEL

This model was proposed by Denning[4] 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 $S_T(t)$ 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 $S_T(t)$, *i.e.*, $lg(S_T(t))$;

(c) $n(t)$, 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:

$$
\begin{cases}
(g_b(w_i p_j, p_t), h_b(w_i p_j, p_t)) & \\
\text{right-hand side of (18a)} & \text{(21a)} \\
\text{right-hand side of (18b), if } w(t,T)=w(t-1,T) & \text{(21b)} \\
(p_t w_i p_j, q_1), \text{ if } p_t \text{ is not in main memory and } w(t,T)>w(t-1,T) & \text{(21c)} \\
(p_t w_t, q_2'), \text{ if } p_t \text{ is not in main memory and } w(t,T)=w(t-1,T) & \text{(21d)} \\
(p_t(w_i-p_t), q_3), \text{ if } p_t \text{ is in } w_t \text{ and } w(t,T)>w(t-1,T) & \text{(21e)} \\
(p_t w_i p_j, q_4), \text{ if } p_t \text{ is not in } W(t-1,T) \text{ but has been tagged and } w(t,T)>w(t-1,T) & \text{(21f)} \\
(p_t w_t, q_5), \text{ if } p_t \text{ is not in } W(t-1,T) \text{ but has been tagged and } w(t,T)=w(t-1,T) & \text{(21g)}
\end{cases}
$$

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 $\#q_\alpha(t)$ and $\#q_\alpha(t,T)$ be the numbers of the q_1 and q_2' 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

$$
\alpha(t,T)=\begin{cases}
\dfrac{\#q_\alpha(t)}{t}, \text{ if } 1\leq t\leq T & \text{(22a)} \\
\dfrac{\#q_\alpha(t,T)}{T}, \text{ if } 1\leq T\leq t & \text{(22b)}
\end{cases}
$$

At $t=0$, the page-turning rate is defined as 0, *i.e.*,

$$\alpha(0,T)=0 \text{ for all } T \qquad \text{(22c)}$$

When $T=0$, the page-turning rate is also defined as 0, *i.e.*,

$$\alpha(t,0)=0 \text{ for all } t \qquad \text{(22d)}$$

By this algorithm, a criterion

$$\frac{b}{T}\geq\alpha(t,T)\geq\frac{a}{T},\ 0<a,b<T \qquad \text{(23)}$$

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 $S_T(t-1)$,

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

(b) When p_t is not in $S_T(t-1)$,

$$
n(t)=\begin{cases}
n(t-1)+1, \text{ if } \alpha(t-1,T)>\dfrac{b}{T} & \text{(24c)} \\
n(t-1), \text{ if } \alpha(t-1,T)\leq\dfrac{b}{T} & \text{(24d)}
\end{cases}
$$

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:

$$1 > \frac{b}{T} > \frac{a}{T} \geq \frac{K(P)}{L_I} \tag{25}$$

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_I is identical to that in [5]. The string for each memory state is represented by a list whose top element is p_t. Each such list is a *LRU stack* as defined by Mattson, *et al*[5]. 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*[5]. The string

Table 1. Illustration of the LRU Algorithm

n	t	0	1	2	3	4	5	6	7	8	9	10	$s(n)$	$f(n)$ $m(n)$	$Nr(n)$ pages p_t recurrent at t
	W_I		a	b	b	c	b	a	d	c	a	a			
2	$Sn(t)$	∧	a	b	b	c	b	a	d	c	a	a	.30	.70	(p_3, p_5, p_{10})
				a	a	b	c	d	a	d	c	c			
	W_o		q_1	q_1	q_0	q_2	q_0	q_2	q_2	q_2	q_2	q_0			
	W_L		a	b		c		a	d	c	a				
	$L(t,n)$		1	2	3	3	4	2	2	2	2	3			
3	$S_n(t)$	∧	a	b	b	c	b	a	d	c	a	a	.50	.50	$(p_3, p_5, p_6, p_9, p_{10})$
				a	a	b	c	b	a	d	c	c			
						a	a	c	b	a	d	a			
	W_o		q_1	q_1	q_0	q_1	q_0	q_0	q_2	q_2	q_0	q_0			
	W_L		a	b		c			d	c					
	$L(t,n)$		1	2	3	4	5	6	3	3	4	5			
4	$S_n(t)$	∧	a	b	b	c	b	a	d	c	a	a	.60	.40	$(p_3, p_5, p_6, p_8, p_9, p_{10})$
				a	a	b	c	b	a	d	c	c			
						a	a	c	c	b	b	b			
	W_o		q_1	q_1	q_0	q_1	q_0	q_0	q_1	q_0	q_0	q_0			
			a	b		c			d						
	$L(t,n)$		1	2	3	4	5	6	7	8	9	10			

or stack updating procedure is imbedded in the recursive definition for the string of a memory state.

As shown by Mattson *et al*[5], 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 $S_n(t)$ is also in $S_{n+k}(t)$. 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 $S_{n+1}(t)$ and $S_n(t)$ implies that

$$lg(S_{n+1}(t)) \geq lg(S_n(t)) \text{ for all } t \tag{26}$$

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

$$L(t, n+1) \geq L(t, n) \text{ for all } t \tag{27}$$

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

$$N_r(n+1) \geq N_r(n) \tag{28}$$

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

$$s_{max} \geq s(n+1) \geq s(n) \geq s(1) \geq 0 \tag{29}$$

The FIFO Algorithm

Belady *et al*[10] first disovered 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 accompained 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*[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

$$s(T) = \frac{N_r(T)}{L_T} \tag{14d}$$

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

$$\Delta(t-1, T) = \begin{cases} 1, \text{ if } h_b(S_T(t-1), p_t) = q_a \text{ or } q_b & \text{(30a)} \\ 0, \text{ otherwise} & \text{(30b)} \end{cases}$$

At $t=1$, the output emitted is always q_1 as shown in (17) so that (30a) holds and yields

$$\Delta(0, T) = 1 \text{ for all } T \tag{30c}$$

At $t=L_I+1$, no page is referenced or the referenced page is $\wedge$ which is the identity of the free monoid P^*. Thus no output is generated or the emitted output is $\wedge$, the identity of the free monoid Q_b^* generated by Q_b since $h_b(S_T(t), \wedge)=\wedge$ for any t holds if the output function h_b is extended such that it maps from $R\times P^*$ into Q_b^*. (30b) implies that

$$\Delta(L_I, T)=0 \text{ for all } T \tag{30d}$$

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

$$m(T)=\frac{1}{L_I}\sum_{t=1}^{L_I}\Delta(t-1, T) \tag{16d}$$

Since the states $S_T(t-1)$ and $S_{T+1}(t)$ are respectively mapped from the substrings $p_{t-T}p_{t-T+1}\cdots\cdots p_{t-1}$ and $p_{t-T}p_{t-T+1}\cdots\cdots p_t$, the following relation which is similar to (3.1) in [11] is established by (30a) and (30b), *i.e.*,

$$w(t, T+1)=w(t-1, T)+\Delta(t-1, T) \tag{31}$$

Define the *average working set size* $w(T)$ for processing the reference string W_I of a finite length L_I as

$$w(T)=\frac{1}{L_I}\sum_{t=1}^{L_I}w(t, T) \tag{32a}$$

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

$$w(0)=0 \tag{32b}$$

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

$$P1:\ 1=w(1)\leq w(T)\leq w(T+1)\leq min(T+1, K(P)) \tag{33}$$

$$P2:\ w(T+1)-w(T)=m(T)-\frac{1}{L_I}w(L_I, T) \tag{34}$$

$$P3:\ 0<f_{min}\leq m(T+1)\leq m(T)\leq m(0)=1 \tag{35}$$

$$P4:\ m(T)=1-s(T) \tag{16b}$$

$$P5:\ m(T+1)-m(T)=-s'(T+1) \tag{36}$$

$$\text{where } s'(T+1)=s(T+1)-s(T) \tag{37}$$

$$P6:\ w(T)=\sum_{z=0}^{T-1}m(z)-\frac{1}{L_I}\sum_{z=0}^{T-1}w(L_I, z) \tag{38}$$

$$P7:\ w(T+1)+w(T-1)\leq 2w(T)-\frac{1}{L_I}[w(L_I, T)-w(L_I, T-1)] \tag{39}$$

$$P9:\ \lim_{T\to L_I} m(T)=f_{min} \tag{40}$$

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

T		t / W_I	0	1	2	3	4	5	6	7	8	9	10	$f(T)$	$s(T)$	$m(T)$	$\sum_{x=0}^{T-1} m(x)$	$w(T)$	$Nr(T)$ pages p_t recurrent at t
1	$S_T(t)$		Λ	a	b	b	c	b	a	d	c	a	a	.40	.20	.80	1.00	1.00	
	W_O			q_1	q'_2	q_0	q'_2	q_5	q_5	q'_2	q_5	q_5	q_0						
	W_L			a	b		c			d									(p_3, p_{10})
	$\Delta(t.T)$		1	1	0	1	1	1	1	1	1	0	0						
2	$S_T(t)$		Λ	a	b	b	c	b	a	d	c	a	a	.40	.30	.70	1.80	1.70	
					a		b	c	b	a	d	c							
	W_O			q_1	q_1	q_3	q_1	q_0	q_5	q'_2	q_5	q_5	q_3						
	W_L			a	b		c			d									(p_3, p_5, p_{10})
	$\Delta(t,T)$		1	1	0	1	0	1	1	1	1	0	0						
3	$S_T(t)$		Λ	a	b	b	c	b	a	d	c	a	a	.40	.40	.60	2.50	2.30	
					a	a	b	c	b	a	d	c	c						
									c	b	a	c							
	W_O			q_1	q_1	q_0	q'_2	q_0	q_4	q'_2	q_5	q_0	q_3						
	W_L			a	b		c			d									(p_3, p_5, p_9, p_{10})
	$\Delta(t,T)$		1	1	0	1	0	1	1	1	0	0	0						
4	$S_T(t)$		Λ	a	b	b	c	b	a	d	c	a	a	.40	.50	.50	3.10	2.70	
					a	a	b	c	b	a	d	c	c						
							a		c	b	a	d	d						
										c	b								
	W_O			q_1	q_1	q_0	q_1	q_3	q_4	q_1	q_0	q_3	q_0						
	W_L			a	b		c			d									$(p_3, p_5, p_8, p_9, p_{10})$
	$\Delta(t,T)$		1	1	0	1	0	1	1	0	0	0	0						
5	$S_T(t)$		Λ	a	b	b	c	b	a	d	c	a	a	.40	.60	.40	3.60	2.90	
					a	a	b	c	b	a	d	c	c						
							a	a	c	b	a	d	d						
										c	b	b							
	W_O			q_1	q_1	q_0	q_1	q_0	q_0	q_1	q_0	q_0	q_3						
	W_L			a	b		c			d									$(p_3, p_5, p_6, p_8, p_9, p_{10})$
	$\Delta(t,T)$		1	1	0	1	0	0	1	0	0	0	0						
6	$S_T(t)$		Λ	a	b	b	c	b	a	d	c	a	a	.40	.60	.40	4.00	3.00	
					a	a	b	c	b	a	d	c	c						
							a	a	c	b	a	d	d						
										c	b	b	b						
	W_O			q_1	q_1	q_0	q_1	q_0	q_0	q_1	q_0	q_0	q_0						
	W_L			a	b		c			d									$(p_3, p_5, p_6, p_8, p_9, p_{10})$
	$\Delta(t,T)$		1	1	0	1	0	0	1	0	0	0	0						

$$P8: \lim_{T \to L_t} w(T) = N_r \tag{41}$$

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

DISCUSIONS

This paper has presetned 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 algoritms 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 prcedure, 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 $S_{n+1}(t-1)=w_ip_j$ be the last transient state and p_t be the current referenced page not in w_ip_j. Then $S_{n+1}(t)=p_tw_ip_j$ by (18c) and $S_n(t)=p_tw_i$ by (18d).

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

$$S_n(t)=\begin{cases}p_t(w_n-p_t), & \text{if } p_t \text{ is in } w_n \quad \text{(A1a)}\\ p_t(w_n-p_k), & \text{if } p_t \text{ is not in } w_n \quad \text{(A1b)}\end{cases}$$

and

$$S_{n+1}(t)=\begin{cases}p_t(w_np_j-p_t), & \text{if } p_t \text{ is in } w_np_j \quad \text{(A2a)}\\ p_tw_n, & \text{if } p_t \text{ is not in } w_n \quad \text{(A2b)}\end{cases}$$

(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 $S_a(t-1)=w_mp_t$ and $S_b(t-1)=w_np_j$ 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_mp_t be also in w_np_j.

(3) p_t be the current referenced page not in w_np_j.

The states $S_a(t-1)$ and $S_b(t-1)$ satisfy the inclusion property because of assumption (2) (c). The state $S_b(t-1)$ being assumed to be steaty in assumption (2) (a) and assumption (3) imply that (19c) or (18d) holds which in turn yields

$$S_b(t)=p_tw_n \quad \text{(B1)}$$

Assumptions (2) (c) and (3) imply that p_t is not in w_mp_t. Since $S_a(t-1)$ is also steady, we have

$$S_a(t)=p_tw_m \quad \text{(B2)}$$

By assumption (2) (b) and (B2), the page p_j which is in $S_a(t)$ does not appear in w_n (since $S_b(t-1)=w_np_j$) or in $S_b(t)$ 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

1. L.A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer," IBM Systems J., Vol. 5, No. 2, 1966, pp. 78-101.
2. B. Randel and C.J. Kuehner, "Dynamic Storage Allocation System," Comm. ACM, Vol. 11, No. 5, May, 1968, pp. 297-305.
3. A.V. Aho, P,J. Denning and J.D. Ullman, "Principles of Optimal Page Replacement," J. ACM, Vol. 18, No. 1, Jan., 1971, pp. 80-93.
4. P.J. Denning, "The Working Set Model for Program Behavior," Comm. ACM, Vol. 11, No. 5, May, 1968, pp.323-333.
5. R.L. Mattson, J. Gecsei, D.R. Slutz and I.L. Traiger, "Evaluation Techniques for Storage Hierachies," IBM Systems J., No. 2, 1970, pp. 78-117.
6. G.S. Shedler and C. Tung, "Locality in Page Reference Strings," IBM Research Report, RJ932, Oct. 29, 1971, pp. 1-48.
7. T.L. Booth, "Sequential Machines and Automata Theory," John Wiley, New York, 1967.
8. E. Gelenbe, "A Unified Approach to the Evaluation of a Class of Replacement Algorithms," IEEE Trans. on Comp., Vol. C-22, No. 6, June, 1966, pp. 611-617.
9. L.A. Belady, "Biased Replacement Algorithms for Multiprogramming," IBM Research Note, NC-697, Yorktown Heights, March 21, 1967, pp. 1-6.
10. L.A. Belady, R.A. Nelson and G.S. Shedler, "An Anomaly in Space-Time Characterisics of Certain Programs Running in a Paging Machine," Comm. ACM, Vol. 12, No. 6, June, 1969. pp. 349-353.
11. P.Denning and S.C. Schwartz, "Properties of the Working-Set Model," Comm. ACM, Vol. 15, No. 3, March, 1972, pp. 191-198.