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

A SELF-BIASED REPLACEMENT ALGORITHM USING THE CHARACTERISTICS OF MARKOVEAN PROGRAM MODEL

WU-HAUNG CHENG
National Chiao Tung University

and

JIUNN HSU
National Tsing Hua University

(Received 1 February 1974)

Abstract: *In this paper a self-biased replacemcet algorithm is developed. The algorithm is mostly suited for allocating system programs and library programs under multiprogramming and/or time-sharing environment. With biasing, the system performance is improved. It is also proved that the algorithm is optimal on macroscopic viewpoint. Since each reference string behaves a static referencing pattern, so it is feasible that the algorithm makes the optimal choice in selecting the candidate for replacement by using the characteristics of Markovean program model. And since in the present time and in the future time, system programs and library programs constitute dorminant part of software system, so it is worthy to develop a replacement algorithm using the characteristics of Markovean program model, although pre-runnings of programs are required.*

INDEX TERMS

Dynamic storage allocation, multiprogramming, time-sharing, convexity, biasing, Markov process.

1. INTRODUCTION

Since many programs (including system programs and users' programs) demand the main memory space concurrently in multiprogramming and/or time-sharing systems, it is rare that a program can reside in the main memory wholly. Only portions of each program (being allocated some memory space) reside in the main memory.

In order that the system be used efficiently, the concept of virtual memory[1,2] has been introduced. The techniques of segmentation and paging[2~6] have been used as the mechanisms for implementing virtual memory, which tackle the relocation problems of program portions (*i.e.*, segments and pages). For allocating program portions, allocation policies are to be introduced. The allocation policies fall into three classes[2,5]:

(1) Replacement policies: They determine which information is to be removed from main memory in order to make room for new allocation.

(2) Fetch policies: They determine when information is to be loaded.

(3) Placement policies: They determine where information is to be placed.

In a paging system, the placement policy for placing K pages is just to use the replacement policy to free K pages[2]. Thus, placement is irrelevant. In this paper, we assume that the system which we consider is a paging system. And we have the information fetches be on demand basis. Thus, replacement is all that is left.

Many storage allocation olgorithms[7~18] have been introduced for solving allocation problems. Before the appearance of the working set model[10] in the literature, processor management and memory allocation had developed and progressed independently. Denning[10] suggested that a unified approach is needed and mentioned that working set brings on the lowest page traffic (*i.e*, the number of pages per unit time being moved between main memory and auxiliary memories) among the four major contenders for page-turning policies (*i.e*, random, first-in/first-out, least recently used, and working set). It was already implemented by Juan Rodriguez-Rossel[19]. Belady mentioned that processor efficiency can be considered as a performance indicator. He presented biased replacement algorithms[9],[12], which favor one of the contending tasks for a period of (p) replacements such that the selected task has all of its pages exempted from consideration by the replacement algorithm within that preriod. This privileged state is then passed on to the other members of the competing set in like fashion. The space allotment of the favored task will not contract, whereas the space allotments of unfavored tasks may contract. Due to the convex relationship between the life time function of task and the alloted space, system performance is improved. Shemer and Gupta[8],[15] presented Bayesian storage allocation algorithms, which use statistical information, such as page reference distribution function[15] and some usage bits, for fetching or replacing a page. Tung[16] developed the concept of apparent contnuity of processing. Under this concept, the working ensemble of a job is saved at the end of last job phase and is retrieved as a whole at the beginning of the next job phase. A job phase is defined such that it can be run without interruption. All the techniques as briefly mentioned above have the same goal: to solve the problem of memery allocation such that processor efficiency is maximized.

In this paper, we shall present a self-biased replacement algorithm using the characteristics of Markovean program model. For "self-biased", we mean that all the resident programs are automatically under biasing. We consider all the systems of variable space allotments to resident programs are under biasing. Since each referencing sting behaves a static referencing pattern, macroscopically. The characteristics of Markovean program model presented in this papae will represent this static referencing pattern. The self-biased replacement algorithm, that we shall present in this paper, uses this static characteristics of Markovean program model for solving the memory allocation problems dynamically such that the system performance is improved. We shall prove that the algorithm is optimal in a sense that process efficiency is improved mostly or, we may say, that the system has the greatest throughput on the average.

In the following, we firstly present some preliminaries such as Markovean

program model, success function, fault function, mean process time, the general convexities, the effects of biasing; then, we present a self-biased replacement algorithm.

2. PRELIMINARIES

In this section we consier some preliminaries for the algorithm which we shall present. They are introduced in the following.

(a) A Markovean Program Model

A general formulation of the mechanism by which program generate reference strings is presented in.[20] which is stated as an l-order nonstationary Markov process. The Markovean program model which we present here is a special case of the one presented in[20], which is considered as an n-order stationary Markov process for an n-page program. The model is stated below as a definition.

Definition 1 A program is a system $M=(N, u_0, f, p)$ is which:

N is set of pages of a program, *i.e.*, $N=\{1, 2, \ldots\ldots, n\}$;

$u_0 \in N$ is the firstly executed page;

f: $N \times N \rightarrow N$ is the page transition map, *i.e.*,

$f(i, j)=j \ \forall \ i.j \in N$;

$p_{ij}=P_r[r_t=j/r_{t-1}=i]$ is the conditional probability that the next reference be to page j given that i is the currently referenced page. (Note: $r_1 \ldots\ldots r_t \ldots\ldots r_T$ is a reference string generated by M.) p_{ij} is independent of time for a stationary Markov process. ※

For the feasibility that the Markov process of program model, which we have just stated, be stationary, we have to discuss real processes in some details. In general, referencing patterns may be classified into four types.[15] They are (1) pure random, (2) pure sequential, (3) random sequential, and (4) sampled sequential search. They suffice to describe a wide variety of software. If a given program has been run for enough number of times, it is feasible that the accumulated data of p_{ij} are independent of time and will represent the type of the referencing patterns of the given program. Especially, we can apply the assumption of stationaryness to system programs and library programs.

Since the occurrence of each state (*i.e.*, a page) of the Markov process, which we have just stated, is almost persistent and aperiodic (except for pure sequential programs), we can also have the assumption of ergodicity for our program model.

Donote P be the matrix of transition probabilities of a stationary ergodic Markov process. That is,

$$P=\begin{pmatrix} p_{11} & p_{12} & \cdots\cdots & p_{1n} \\ p_{21} & p_{22} & \cdots\cdots & p_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ p_{n1} & p_{n2} & \cdots\cdots & p_{nn} \end{pmatrix} \qquad (1)$$

And B is row vector, *i.e.*, $B=[b_1, b_2, \ldots\ldots b_n]$, such that

$$B \times P = B \qquad (2)$$

That is, b_i is the long-run probability that a reference is to page i.

(b) Success Function, Fault Function, and Mean Process Time

Suppose that the space allotment for a given program at time t is $q(t)$. Let $Q_{q(t)}$ be the set of the pages resided in the main memory at time t. That is,

$$Q_{q(t)}=\{K_i/K_i\in N,\ 1\leq i\leq q(t)\} \tag{3}$$

It can be seen, from Eq. (3) that $Q_{q(t)}$ is a subset of N, $Q_{q(t)}\subseteq N$. Hence, $N-Q_{q(t)}$ denotes the set of those pages which are not resided in the main memory.

For a reference to be successful, it means that given a reference to a residing page the next reference is also to a residing page. According to this notion, we define the "success function" of a process as

$$S_{q(t)}=P_r[r_{t+1}\in Q_{q(t)}/r_t\in Q_{q(t)}] \tag{4}$$

Eq. (4) can be restated in terms of transition probabilities and long-run probabilities. That is,

$$S_{q(t)}=\sum_{s=1}^{q(t)}\frac{b_{k_s}}{c}\left(\sum_{u=1}^{q(t)}P_r[K_u/K_s]\right), \tag{5}$$

where

$$C=\sum_{m=1}^{q(t)}b_{k_m} \tag{6}$$

In Eq. (5), $\frac{b_{k_s}}{c}$ is the relative frequency of referencing page K_s to referencing $Q_{q(t)}$, where $K_s\in Q_{q(t)}$. And $\{\sum_{u=1}^{q(t)}p_r[k_u/k_s]\}$ is the conditional probabbility of the successfulness of the next reference given the present reference is to page K_s.

Having derived the "success function", the "fault function" can be stated as

$$F_{q(t)}=P_r[r_{t+1}\in N-Q_{q(t)}/r_t\in Q_{q(t)}] \tag{7}$$

$$=1-S_{q(t)} \tag{8}$$

It can be easily understood that $F_{q(t)}$ is the conditional probability that the next reference is outside $Q_{q(t)}$ given that the present reference is successful. In this case, a page fault occurs.

The mean recurrence time of page fault is

$$\bar{m}_{q(t)}=\frac{\triangle}{F_{q(t)}}=\frac{\triangle}{1-S_{q(t)}}, \tag{9}$$

where $\triangle$ is the average reference time.[1] $\bar{m}_{q(t)}$ can also be considered as the mean process time. Eq. (9) comes from a result of Markov Theory.[21]

(c) The General Convexities of $S_{q(t)}$, and $\bar{m}_{q(t)}$

From Eq. (9), we have the functional relationships among $S_{q(t)}$, $F_{q(t)}$, and $\bar{m}_{q(t)}$. Now, we discuss what characteristics shall be implied by Eq. (9).

Definition 2 An ordered set of resident pages with $q(t)$ as the space allotment for the corresponding program at time t is

$$Q^o_{q(t)}=\{K_i/K_i\in N,\ b_{k_i}\geq b_{k_{i+1}},\ 1\leq i\leq q(t).\} \tag{10}$$※

$Q^o_{q(t)}$ is much the same as $Q_{q(t)}$ (cf. Eq. (3) but with its elements being ordered according to their long-run probabilites. From $Q^o_{q(t)}$, we can easily grasp the order of the activities of resident pages of a given program. With $Q^o_{q(t)}$ in mind, we now discuss the convex characteristic of $S_{(t)}$. It is stated below as a theorem.

(In the sequel, we assume that all quantities are time-invariant. So, t is no longer a parameter.)

Theorem 1. Let Q^o_{q-1}, Q^o_q, and Q^o_{q+1} be ordered sets of resident pages for a given program such that $Q^o_{q-1} \subseteq Q^o_q \subseteq Q^o_{q+1}$; and let S_{q-1}, S_q, and S_{q+1} be their corresponding success functions respectively. Then,

$$S_{q+1}-S_q \leq S_q-S_{q-1} \tag{11}$$

(the proof of Theorem 1 is given in the Appendix.) ※

Eq. (11) implies that S_q is a monotonically increased convex-up function of q. Without no loss of generality we have the function be continuous. It is shown by Fig. 1.

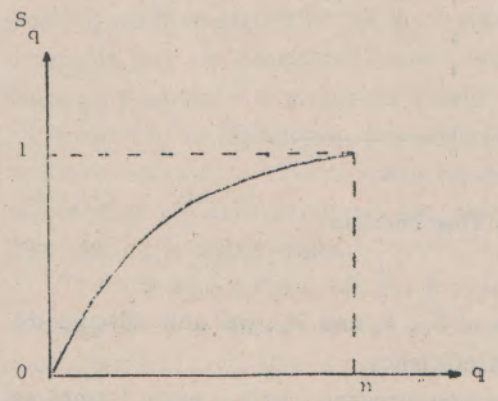

Fig. 1. Success Function

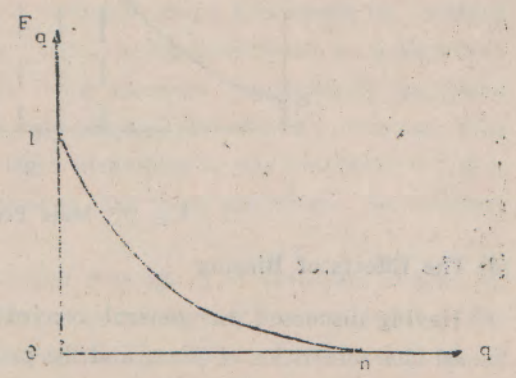

Fig. 2. Fault Function

From Eq. (9) and Eq. (11), we can easily obtain the convex characteristics of F_q and $\overline{m}_q$. They are given in the following.

$$F_{q-1}-F_q \geq F_q-F_{q+1} \tag{12}$$

$$\overline{m}_{q+1}-\overline{m}_q \geq \overline{m}_q-\overline{m}_{q-1} \tag{13}$$

Eq. (12) implies that F_q is a monotonically decreased convex-down function of q. And Eq. (13) implies that $\overline{m}_q$ is a monotonically increased convex-down function of q. Similarly as we treat S_q, without no loss of generality we have the functions of F_q and $\overline{m}_q$ be continuous. They are shown by Fig. 2 and Fig. 3 respectively.

Now, we discuss some marginal characteristics of S_q, F_q, ann $\overline{m}_q$:

(i) For $q=0$, we have $S_0=0$, $F_0=1$, and $\overline{m}=\triangle$. This means that a page fault occurs for each reference, and the mean procss time is equal to the average reference time.

(ii) For $q=n$, we have $S_n=1$, $F_n=0$, and $\overline{m}_n=\infty$. This means that each reference succeeds with probability one, no page fault shall occur for each reference. And the given program can be run indefinitely without any interruption; so, in principle, the program may be executed many, many times according to the programmer's will.

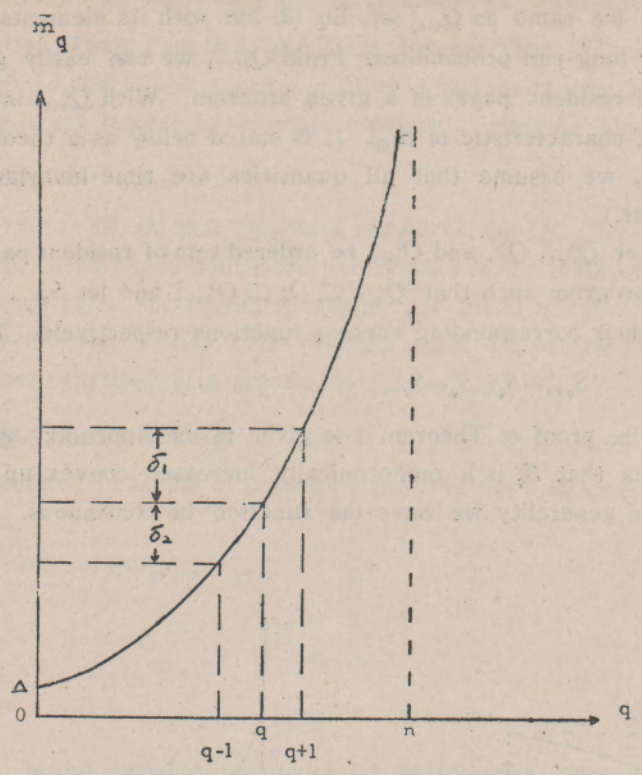

Fig. 3. Mean Process Time Function

(d) The Effects of Biasing

Having discussed the general convexities of S_q, F_q and $\bar{m}_q$, we now discuss the biased characteristics of them and the process efficiency.

Definition 3. The process efficiency of a given program, with q page frames as its space allotment, is given by

$$\eta_q = \frac{\bar{m}_q}{\bar{m}_q + d} \tag{14}$$

where d is the expected system overhead for handling the interruption caused by a page fault.

(We assume that d is a constant.)

Let

$$\bar{m}_{q+1} - \bar{m}_q = \delta_1 \tag{15}$$

$$\bar{m}_q - \bar{m}_{q-1} = \delta_2 \tag{16}$$

From Eq. (9) and Eq. (12), we have

$$F_{q+1} + F_{q-1} - 2F_q = -\frac{[2\delta_1\delta_2 - \bar{m}_q(\delta_1 - \delta_2)] \cdot \Delta}{\bar{m}_q(\bar{m}_q + \delta_1)(\bar{m}_q - \delta_2)} \geq 0 \tag{17}$$

Thus,

$$\beta = 2\delta_1\delta_2 - \bar{m}_q(\delta_1 - \delta_2) \geq 0 \tag{18}$$

Eq. (18) states the relationships among δ_1, δ, and $\bar{m}_q$.

Under the fact stated by Eq. (18), we now discuss what happens to the process efficiency. From Eq. (14), (15), and (16), we have

$$\eta_{q+1}+\eta_{q-1}-2\eta_q = \frac{(\bar{m}_q+d)(\delta_1-\delta_2)-2\delta_1\delta_2}{(\bar{m}_q+\delta_1+d)(\bar{m}_q-\delta_2+d)(\bar{m}_q+d)} \tag{19}$$

$$= \frac{d(\delta_1-\delta_2)-\beta}{(\bar{m}_q+\delta_1+d)(\bar{m}_q-\delta_2+d)(\bar{m}_q+d)} \tag{20}$$

In order that the average process efficiency under biasing is improved, we must have

$$d(\delta_1-_2)\geq\beta \tag{21}$$

Otherwise, biasing will not improve process efficiency.

If there is no overhead for handling the interruption caused by a page fault (*i.e.*, $d=0$), then it is impossible that the process efficiency can be improved under biasing. But since d is not equal to zero, so we can hope that the process efficiency can be improved under biasing, if Eq. (21) is satisfied. According to Belady's simulaection,[12] the technique of space biasing does improve the system performance. In general, we may state that all the systems of variable space allotments to resident programs may be considered under biasing (with q being considered as a variable). Since only portions of programs reside in the main memory concurrently, the space allotment for each program may sometimes expands and sometimes contracts. This evidence indicates a kind of space biasing. And according to the results of[18,22], it is stated that variable partitions are more efficient than fixed partitions. So we state that biasing is worth while.

There is also a drawback for a system under biasing. The drawback is that the average number of page faults is increased, compared with a system which is not under biasing. But this is not the problem since the operations of processor and I/0 channels are in parallel, what we want to do is to increase the average process efficiency, that is, to increase the average throughput.

3. A SELF-BIASED REPLACEMENT ALGORITHM

In this section we shall present a self-biased replacement algorithm which uses the characteristics of Markovean program model presented in the previous section.

We assume that the main memory is in full-loaded environment. So, whenever a page fault occurs, there must be a currently resident page chosen for replacement in order to make a page frame for the incoming page.

Suppose there are portions of h programs reside in the main memory concurrently. Let them be N^1, N^2,....., N^h such that

$$N^l=\{1^l, 2^l, \ldots\ldots, n^l\},\ 1\leq l\leq h \tag{22}$$

And so forth, we have

P^l : the matsix of transition probabilities of N^l.

B^l : the vector of long-run probabilities of N^l.

$Q^l_{q(t)}=\{K_t{}^l/K_t{}^l\in N^l,\ 1\leq i\leq q^l_{(t)}\}$

: the set of resident pages of N^l.

From the above statements we have the success function of N^l as

$$S^l_{q(t)}=\sum_{s=1}^{q^l_{(t)}}\frac{b^l_{k_s}}{c^l}\left(\sum_{u=1}^{q^l_{(t)}}P_r[K^l_u/K^l_s]\right)=P_r[r^l_{t+1}\in Q^l_{q(t)}/r^l_{t-1}\in Q^l_{q(t)}] \tag{23}$$

where

$$C^l\triangleq\sum_{m=1}^{q^l_{(t)}}b^l_{k_m} \tag{24}$$

When a page fault occurs, one of the resident pages must be chosen for replacement since the main memory is full-looaded. Replacement algorithm will choose a candidate according to its policy. The replacement policy which we shall present is stated in the following.

Let r^l be the currently executed page of N^l, or the temporarily suspended page of N^l waiting for some pending activity such as a signal from another process, an $I/0$ interrupt, a page-in or -out, or a exteral interrupt. It is evident that (r^l, $1\leq l\leq h$) must not be chosen as the candidate for replacement. The candidate must be the one with the least activity in the near future.

Suppose that

$$P_r[m^l/r^l]=\min_{\substack{K^l_i\in Q^l_{q(t)}\\ K^l_i\neq r^l}}P_r[K^l_i/r^l],\quad 1\leq l\leq h \tag{25}$$

Eq. (25) means that the next reference is to m^l has minimal transition probability among resident pages of N^l given that the current reference is to r^l. So, whenever we want to choose a page from N^l for replacement, m^l is the candidate.

Having find the replacement candidate for each process, we now state the criterion for selecting which process to be the candidate of removing a page from it. We firstly define the marginal process efficiency of N^l.

Definition 4. Let $S^l_{q(t)}$, $\bar{m}^l_{q(t)}$, and $\eta^l_{q(t)}$ be the success function, the mean process time, and the process efficiency of N^l, respectively, with $Q^l_{q(t)}$ as its set of resident pages. And let $S^l_{m,q(t)}$, $\bar{m}^l_{m,q(t)}$, and $\eta^l_{m,q(t)}$ be the success function, the mean process time, and the process efficiency of N^l, respectively, with $Q_{q(t)}-\{m^l\}$ as its set of resident pages. Thus, we have

$$\bar{m}^l_{q(t)}=\frac{\Delta}{1-S^l_{q(t)}} \tag{26}$$

$$\eta^l_{q(t)}=\frac{\bar{m}^l_{q(t)}}{\bar{m}^l_{q(t)}+d} \tag{27}$$

$$\bar{m}^l_{m,q(t)}=\frac{\Delta}{1-S^l_{m,q(t)}} \tag{28}$$

$$\eta^l_{m,q(t)}=\frac{\bar{m}^l_{m,q(t)}}{m^l_{m,q(t)}+d} \tag{29}$$

The marginal process efficiency is defined as

$$\Phi_{q(t)}=\eta^l_{q,(t)}-\eta^l_{m,q(t)} \tag{30}$$

With the marginal process of $N^l(1 \leq l \leq h)$ in mind, the process with least marginal process efficiency is chosen as the candidate of removing a page from it. It is stated below as a theorem.

Theorem 2. It is optimal that the process with the least marginal process efficiency is chosen as the condidate of removing a page from it.

〔Proof〕 If we choose the process with the least marginal process efficiency as the candidate of removing a page from it, then the decrease of process efficiency due to replacement is minimal. Q.E.D. ※

The algorithm is shown by Fig. 4. The reason of why it is called "self-biased" is that the biasing is done automatically and dynamically.

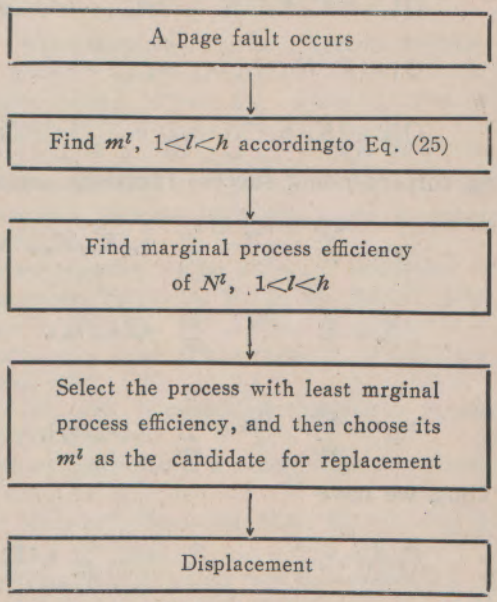

Fig. 4. A self-biased replacement algorithm

4. DISCUSSIONS AND CONCLUSIONS

In this paper we have developed a self-biased replacement algorithm using the characteristics of Markovean program model. It is mostly suited for allocating system programs and library programs. Since they are run very frequently.

On macroscopic viewpoint, each referencing string behaves a static referencing pattern, and this is what we present is this paper. The success function of a program in execution may be considered as a dynamic locality parameter. With the self-biased replacement algorithm presented in this paper, we can expect the locality of the referencing pattern in the near future by using the statistic of previous runnings. And we have proved that the biased replacement algorithm is optimal, macroscopically.

We also mentioned in the previous section that the systems of variable space allotments to resident programs may be considered under biasing. So we conclude that the systems with variable partitions are more efficient than that with fixed partitions. The same statements can also be found in 〔22〕. Although the average number of page faults is increased, the average process efficiency is improved, so that the system has a greater throughput.

We can also apply the characteristics of Markovean program model to scheduling problem. That is, whenever a process gains the control of a process, we may let the alloted time slice be its current mean process time. Or, we may let the alloted time slice be the cycle time multiplied by the ratio of the indicated mean process time to the sum of mean process times of all active programs, if the cycle time, should be distributed to all active programs.

In the present time and in the future time, system programs and library programs

constitute dorminant part of software systems. So it is worthy and feasible to develop a reglacement algorithm using the characteristics of Markovean program model.

APPENDIX

Proof of Theorem 1:

Since we have

$$Q^0_{q-1}=\{K_i/K_i\in N,\ b_{k_i}\geq b_{k_{i+1}},\ 1\leq i\leq q-1\} \tag{1}'$$

$$Q^0_q=\{K_i/K_i\in N,\ b_{k_i}\geq b_{k_{i+1}},\ 1\leq i\leq q\} \tag{2}'$$

$$Q^0_{q+1}=\{K_i/K_i\in N,\ b_{k_i}\geq b_{k_{i+1}},\ 1\leq i\leq q+1\} \tag{3}'$$

the corresponding success functions are given as

$$S_{q-1}=\sum_{s=1}^{q-1}\frac{b_{k_s}}{c_{-1}}\sum_{u=1}^{q-1}p_r[K_u/K_s],\quad \text{where}\quad c_{-1}=\sum_{j=1}^{q-1}b_{k_j} \tag{4}'$$

$$S_q=\sum_{s=1}^{q}\frac{b_{k_s}}{c_0}\sum_{u=1}^{q}p_r[k_u/k_s],\quad \text{where}\quad c_0=\sum_{j=1}^{q}b_{k_j} \tag{5}'$$

$$S_{q+1}=\sum_{s=1}^{q+1}\frac{b_{k_s}}{c_1}\sum_{u=1}^{q+1}p_r[k_u/k_s],\quad \text{where}\quad c_1=\sum_{j=1}^{q+1}b_{k_j} \tag{6}'$$

Thus, we have

$$C_0(S_q-S_{q-1})=-b_{k_q}S_{q-1}+b_{k_q}\sum_{u=1}^{q-1}p_r[k_u/k_q]+\sum_{s=1}^{q}b_{k_s}p_r[k_q/k_s] \tag{7}'$$

$$C_0(S_{q+1}-S_q)=-b_{k_{q+1}}S_q+b_{k_{q+1}}\sum_{u=1}^{q}p_r[k_u/k_{q+1}]+\sum_{s=1}^{q+1}b_{k_s}p_r[k_{q+1}/k_s] \tag{8}'$$

Since

$$b_{k_q}\ S_{q+1}<b_{k_q+1}\ S_q \tag{9}'$$

$$b_{k_q}\sum_{u=1}^{q-1}p_r[k_u/k_q]>b_{k_{q+1}}\sum_{u=1}^{q}p_r[k_u/k_{q+1}] \tag{10}'$$

$$\sum_{s=1}^{q}b_{k_s}\ p_r[k_q/k_s]>\sum_{s=1}^{q+1}b_{k_s}\ p_r[k_{q+1}/k_s] \tag{11}'$$

(Note that Eq. (9)′, (10)′, and (11)′ hold on the basis of approximation.)

Thus, we complete the proof. The same statement can be found in [9, 12, 15]. ※

REFERENCES

1. T. Kilburn, D.B.G. Edwards, M.J. Lanigan, and F.H. Sumner, "One-level storage system", IRE Trans. on Computers, Vol. EC-11, No. 2, pp. 223-235, April., 1962.
2. P.J. Denning, "Virtual memory", Computing Sureys, Vol. 2, No. 3, pp. 153-189, Sep., 1970.
3. J.B. Dennis, "Segmentation and the design of multi-programmed computer systems", J. of ACM, Vol. 12, No. 4, pp. 589-602, Oct., 1965.
4. W. Arden, B.A. Galler, T.C. O'Brien, and F.H. Westervelt, "Progrrm and addressing structure in a timesharing euvironment", J. of ACM, Vol. 13, No. 1, pp. 1-16. Jan., 1966.

5. B. Randell and C.J. Kuehner, "Dynamic storage allocation systems", Comm. of ACM, Vol. 11, No. 5, pp. 297-306, May, 1968.
6. J.L. Smith, "Multiprogramming under a page on demand strategy", Comm. of ACM, Vol. 10, No. 10, pp. 636-645, Oct., 1967.
7. L.A. Belady, "A study of replacement algorithms for a virtual storage", IBM Systems Journal. Vol. 5, No. 2, pp. 77-101, 1966.
8. J E. Shemer ard G. A. Seippry, "Statistical analysis of paged and segmented computer systems", IEEE Trans. on Computers, Vol EC-15, No 6, pp. 855-864, Dec., 1966.
9. L. A. Belady, "Biased replacement algoithms for multiprogramming", Rep. NC 697, IBM Thomas J. Watson Res. Center Yorktown Heights, N.Y., March, 1967.
10. P.J. Denning, "The working set model for program behavior", Comm. of ACM, Vol. 11, NO. 5, pp. 323-333, May. 1968.
11. P.J. Denning and S.C. Schwartz, "Properties of the working-set model", Comm. of ACM, Vol. 15, No. 3, pp. 191-198, March 1972.
12. L.A. Belady and C.J. Kuehner, "Dynamic space-sharing in computer systems", Comm. of ACM, Vol. 12, No. 5, pp. 282-288, May, 1969.
13. P.J. Denning, "Equipment configuration in balanced computer systems", IEEE Trans. on Computer, Vol. C-18, No. 11, pp. 1008-1012, Nov., 1969.
14. M.V. Wilkes, "A model for core space allocation in a time-sharing system", AFIPS, SJCC, pp. 265-271, 1969.
15. J.E. Shemer and S.C. Gupta, "on the design of Bayesian storage allocation olgorithms for paging and segmentation", IEEE Trans. on Computers, Vol. C-18, No. 7, pp. 644-751, July, L969.
16. Chin Tung, "On the apparent continuity of processingin a paging environment", IEEE Trans. on Computers, Vol. C-19, No. 11, pp. 1047-1054, Nov., 1970.
17. R.L. Mattson, J. Gecsei, D.R. Slutz, and I.L. Traiger, "Evaluation techniques for storage hierarchies", IBM Systhm J. 9, No. 2, pp. 78-117, 1970.
18. P.H. Oden and G.S. Shedler, "A model of memory contention in a paging machine", Comm. of ACM, Vol. 15, No. 8, pp. 761-771, Aug. 1972.
19. Juan Rodriguez-Rossel, "Empirical working set behavior", Comm. of ACM, Vol. 16, No. 9, pp. 556-560, Sep. 1973.
20. A.V. Aho., P.J. Denning, and J.D. Ullman, "Principles of optimal page replacement", J. of ACM, Vol. 18, No. 1, pp. 80-93, Jan., 1971.
21. William Feller, "An intorduction to probability theory and its applications", Vol. 1, 3rd edition, 1968, pp. 309.
22. E.G. Coffman Jr. and T.A. Ryan Jr., "A study of storage partitioning using a mathematical model of locality", Comm. of ACM, Vol. 15, No. 3, pp. 185-190, March 1973.