Performance Analyses of Paging Algorithms for Compilation of a Highly Modularized Program

TAKESHI CHUSHO AND TOSHIHIRO HAYASHI

Abstract—Previous works on paging behavior have mainly concentrated on procedures, not on data. This paper is an attempt to clarify the paging behavior of data referenced by a newly developed language processor, and theoretically analyze the performance of several page replacement algorithms with no loss of generality.

The experimental analyses show that the optimum page size for these data is small and locality is evident but not so high in comparison with that for procedures, and that the efficiency of each replacement algorithm can be ranked in descending order as LRU, simplified LRU's, and FIFO. On the basis of these results, the language processor is improved.

The performance of algorithms is theoretically analyzed assuming the existence of locality. The difference in performance between LRU and FIFO is evaluated by the upper and lower bound functions and proves to increase at low fault rate. The reason for the difference between LRU and simplified LRU's is analyzed by the period measure for which information about recent references to pages is collected. Thus, it is demonstrated that the performance of simplified LRU's is consistent with this measure and strongly depends on the reset timing of LRU-flags. These theoretical results are not limited to compilation only.

Index Terms—Compilation, fault rate, locality, LRU, modular programming, optimum page size, paging behavior of data, relative difference in performance, replacement algorithm.

Manuscript received July 25, 1979; revised March 18, 1980.

T. Chusho is with the Systems Development Laboratory, Hitachi, Ltd., Kawasaki, Japan.
T. Hayashi is with Omika Works, Hitachi, Ltd., Ibaraki, Japan.

I. Introduction

RECENT new programming methodologies such as stepwise refinement [1], [2] and data abstraction [3] make a program highly modularized [4]. To compile such a module separately, a processor must frequently refer to a lot of data including information about other related modules. Therefore, paging between main and auxiliary memories is required for these data.

There are two problems, however, when applying previous studies [5]-[10] on page replacement algorithms to this case. Firstly, most experimental analyses of paging behavior have been performed on programs or procedures, not exclusively on data, and almost nothing is known about the paging behavior of data. Secondly, comparison between practical replacement algorithms is based on empirical results and theoretical approaches have not yet been sufficiently researched.

With regard to the paging behavior of programs, Belady [6] pointed out the nonlinear property that the average execution interval was approximated by as^k at small memory capacity, where a was constant for an individual program, s was the memory capacity, and $k \approx 2$. As for data, on the one hand, the relational database machine RAP [11] was designed assuming the existence of high locality. On the other hand, empirical studies [12], [13] indicated the lack of locality in the hierarchical database systems.

This paper analyzes the paging behavior of various tables referenced by the compiler of SPL [14] supporting the aforementioned modularization techniques, and clarifies characteristics of data references with respect to optimum page size, locality, and effect of several replacement algorithms.

Many replacement algorithms have been studied together under reference strings and mathematical analyses have also been done [9]. For example, Mattson [7] formalized the stack algorithm such as LRU. Some theoretical analyses, however, remain to be done with respect to the difference in performance between representative algorithms such as LRU, simplified LRU's, and FIFO. Although LRU and FIFO have been investigated and compared in most experiments on paging systems, their relation is not yet sufficiently known because it is difficult to mathematically relate locality to the FIFO algorithm. In this paper, the difference in performance between LRU and FIFO is evaluated by the upper and lower bound functions assuming the existence of locality.

Furthermore, simplified LRU's are considered because these algorithms are of practical use. They use LRU-flags instead of an LRU stack and some computers support the manipulation of these flags in hardware. However, full LRU is not practical because of stack manipulation which requires extended time, although this algorithm is suitable for references with locality. Therefore, the reason for the difference between LRU and simplified LRU's is investigated by the period measure for which information about recent references to pages is collected. The relation between the reset timing of LRU-flags and the performance of simplified LRU's is discussed on the basis of this analysis.

II. Compilation of a Modularized Program

The newly developed language, SPL [14], provides ample and powerful facilities for modularization, top-down development by stepwise refinement, data abstraction by encapsulation of a data type and its operations, and structured coding.

One of the main original features is that a program hierarchy is represented by separating declarations of common data from procedures. That is, in SPL, there are two types of modules, namely, an environment module and a process module. The environment module is composed of declarations of variables, constants, data types, etc., and constructs a tree structure as shown in Fig. 1. The process module is a set of procedures and is positioned under a suitable environment module as its descendant.

For separate compilation of such a modularized program, intermediate results of upper modules are stored in a library

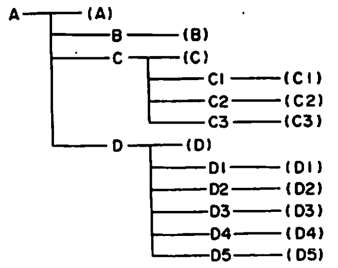

Fig. 1. Example of a hierarchically modularized program.

and are referenced at the time their descendant modules are compiled. Therefore, compiling speed strongly depends on the frequency of paging between main and auxiliary memories for such data. In the next chapter, the paging behavior of these data is analyzed.

III. Experimental Analyses of Paging Behavior

A. Method

Reference strings to data were stored in an auxiliary memory while compiling the modularized program shown in Fig. 1, and were analyzed later. These reference strings were partitioned into the following three parts:

1) analysis of environment modules (ENV),
2) analysis of process modules (PROC),
3) generation of an object program (GEN).

The items of analyses are

1) frequency of reference to each depth of an LRU stack and to each page in the virtual space; and
2) fault rates by application of such replacement algorithms as LRU, FINUFO, FIFO, and FIVE.

LRU and FIFO are well-known. FINUFO (first in not used first out) and FIVE are simplified LRU's and use LRU-flags of 1 bit and 5 bits, respectively, instead of an LRU stack.

In FINUFO, every page in the buffer has an LRU-flag which is set at the time of reference to the corresponding page. Some computers support this mechanism in hardware. At a page fault, the buffer is cyclically searched for a page with a reset flag starting from the page after the page loaded last, and the page found first is paged out. During this search, the set flags are reset.

In FIVE, every page in the buffer has an LRU-flag of 5 bits and the leftmost bit is set when the corresponding page is referenced. At a page fault, the page which flag has the minimum integer value is paged out and every flag is divided by 2.

B. Page Size

Fig. 2 shows fault rates of LRU for page sizes 128, 256, and 512 bytes. These results show that a smaller page size is better under the fixed capacity of the buffer.

C. Locality

Locality is defined using the probability $L(i)$ of reference to the depth i of an LRU stack as follows:

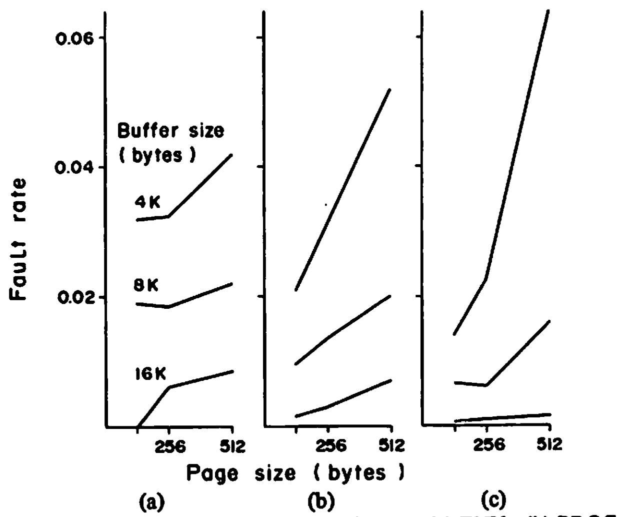

Fig. 2. Relation between page size and fault rate. (a) ENV. (b) PROC. (c) GEN.

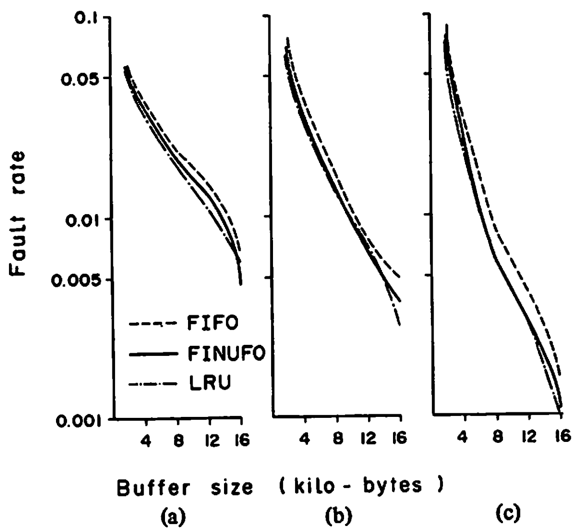

Fig. 3. Performance of paging algorithms. (a) ENV. (b) PROC. (c) GEN.

TABLE I
FREQUENCY OF REFERENCE* TO EACH DEPTH IN AN LRU STACK AND TO EACH PAGE IN VIRTUAL SPACE

(a) in an LRU stack			(b) in virtual space		
depth	freq.	rate(%)	order	freq.	rate(%)
1	18164	60.5	1	4903	16.3
2	5360	17.9	2	2271	7.6
3	1267	4.2	3	2148	7.2
4	1263	4.2	4	2055	6.9
5	1034	3.4	5	1616	5.4
6	447	1.5	6	1527	5.1
7	347	1.2	7	1036	3.5
8	236	0.8	8	783	2.6
9	212	0.7	9	682	2.3
10	127	0.4	10	572	1.9

*First 30 000 data references of PROC-job with 256 bytes/page.

$$L(i) > L(j), \quad i < j. \tag{1}$$

Observing the frequency distribution in the LRU stack as shown in Table I-a, the high locality is indicated by the fact that the frequency of reference to the top page is greater than 60 percent and the frequency of each page rapidly decreases according to the depth of the stack.

D. Static Frequency Distribution

The frequency of reference to each page in the virtual space was also measured and the $b - 1$ pages of the highest frequency were excluded from paging. Such a partially preloaded method [10], however, was less effective than LRU in our experiments. The reason for this seems to be that the frequency distribution in the LRU stack is sharper than that in the virtual space as shown in Table I-a and b.

E. Performance of Replacement Algorithms

The four algorithms mentioned above were applied to the reference strings and fault rates were observed. In almost all cases, good performance was obtained in the descending order of LRU, FINUFO, FIVE, and FIFO. Fig. 3 presents part of the results. Fig. 4 plots the relative difference in performance for each algorithm to LRU, that is,

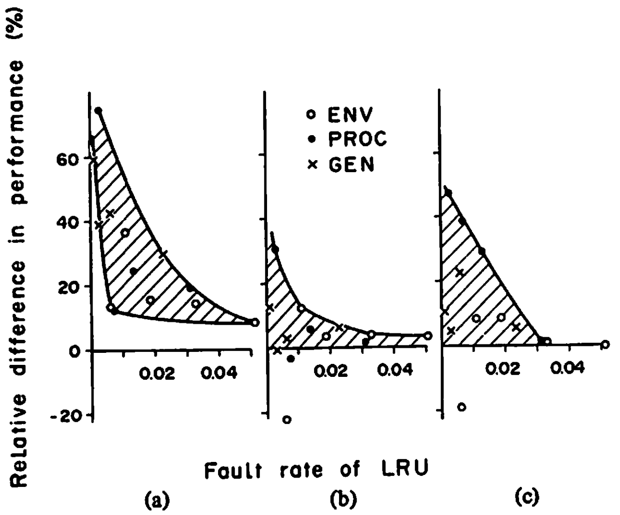

Fig. 4. Relative difference in performance for FIFO and simplified LRU's to LRU. (a) FIFO to LRU. (b) FINUFO to LRU. (c) FIVE to LRU.

$$g(f_{\text{LRU}}) = \frac{\text{fault rate of each algorithm}}{\text{fault rate of LRU}} - 1. \tag{2}$$

The following observations are made from these results.

1) FIFO is evidently less effective, especially at low fault rate.

2) FINUFO shows comparatively stable performance and the degradation to LRU is less than 15 percent.

3) FIVE is almost the same as FINUFO but is less effective at low fault rate.

IV. Theoretical Analyses of Results

A. Optimum Page Size

Our experiments demonstrated that a smaller page size was better as shown in Fig. 2. The reason for this seems to be that the probability of reference to the same data as the last referenced data is much higher than the probability of reference to neighboring data. That is, let $p(y/x)$ denote the probability that the location y is referenced directly after the location x;

and the characteristic of data references in compilation is represented as follows:

$$p(x/x) \gg p(x \pm d_1/x) > p(x \pm d_2/x), \quad 0 < d_1 \ll d_2, \tag{3}$$

in contrast with procedures in which optimum page size is rather large because the probability of reference to a neighbor is higher than that to the same instruction.

Optimum page size, however, cannot be determined from Fig. 2 because the mapping table enlarges in inverse proportion to page size [8]. The remainder of this section describes how to determine the optimum page size.

When the size of the referenced data area is D and the page size is s, the size of the mapping table with entry length a is $a(D/s)$. This table shares with a buffer of the size B in an available main memory of the capacity C, that is,

$$B + a(D/s) = C. \tag{4}$$

Meanwhile, I/O time per one page between main and auxiliary memories is expressed as $u + s/v$ by the average access time u and the data transfer speed v, and thus I/O time t per reference is obtained as follows:

$$t(s,B) = (1 + w)(u + s/v) f(s,B) \tag{5}$$

where w is the write ratio and f is the fault rate.

Consequently, the optimum page size will be s which minimizes t under the constraint condition (4). In our experiments, the actual data of $f(s,B)$ shown in Fig. 2 were used to solve (5) and the optimum page size was obtained as 256 bytes.

B. The Relation Between Buffer Size and Fault Rate

The results shown in Fig. 3 were analyzed in detail and the relation between buffer size and fault rate was approximated as follows:

$$f(b) = Gb^{-k}, \quad f \geqslant f_0 \tag{6}$$

$$f(b) = He^{-\alpha b}, \quad f < f_0 \tag{7}$$

where b is the number of buffer pages and values of parameter sets (G, k, H, α, f_0) are, respectively,

$$\begin{aligned} &\text{ENV:} \quad (0.14, 0.48, 0.058, 0.035, 0.04),\\ &\text{PROC:} \quad (0.40, 0.90, 0.068, 0.050, 0.03),\\ &\text{GEN:} \quad (0.34, 0.58, 0.065, 0.066, 0.18). \end{aligned} \tag{8}$$

Equations (6) and (7) are observed from Fig. 5 and Fig. 3, respectively.

The study mentioned previously [6] also reported that (6) was true at the small memory capacity. The values of k, however, are different, that is, $k < 1$ in our experiment but $k \approx 2$ in the previous work. This is because locality of data references in compilation is not so high as locality of procedures. This property has been observed in the paging behavior of a Snobol compiler by Coffman [5].

C. The Comparison of FIFO with LRU

Theoretical analyses of replacement algorithms have been conducted in previous works [6]-[10]. Comparison of FIFO with LRU, however, has not yet been theoretically analyzed because it is difficult to mathematically relate locality to the FIFO algorithm. The remainder of this section will prove, by evaluating the upper bound and the lower bound, that the relative difference in performance between the algorithms increases at low fault rate. This result will be general except for the assumption of locality.

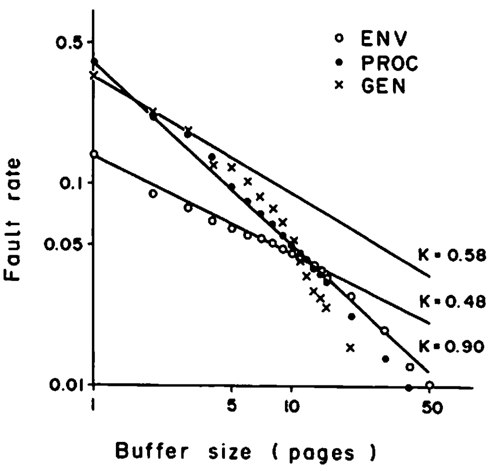

Fig. 5. Performance of LRU at small memory capacity.

Let LRU and FIFO be applied to the same reference strings and let G_{LRU} and G_{FIFO} denote sets of pages in the buffer, respectively. The relation between G_{LRU} and G_{FIFO} is given by

$$G_{\text{FIFO}} = G_{\text{LRU}} - \{U_1, \cdots, U_k\} + \{V_1, \cdots, V_k\}, \quad 0 \leqslant k < b \tag{9}$$

where U_i is only in G_{LRU} and V_i is only in G_{FIFO}.

If a page fault occurs, the bth page in the LRU stack should be paged out in LRU and the page q loaded first should be paged out in FIFO. The difference d of both fault rates is equal to the difference in the summations for the reference probability p of each page in the buffer as follows:

$$d = f_{\text{FIFO}} - f_{\text{LRU}} = \sum_{i=1}^{k} p(U_i) - \sum_{i=1}^{k} p(V_i) + p(q) - L(b+1). \tag{10}$$

Since q is selected without regard to the position in the LRU stack, $p(q)$ must be the average of G_{FIFO} defined in (9) as follows:

$$p(q) = \frac{1}{b}\left\{\sum_{i=2}^{b+1} L(i) - \sum_{i=1}^{k} p(U_i) + \sum_{i=1}^{k} p(V_i)\right\}. \tag{11}$$

Meanwhile, (1) implies

$$p(U_i) > p(V_j), \quad 1 \leqslant i, j \leqslant k. \tag{12}$$

Therefore, substituting (11) and (12) into (10), the lower bound of d is

$$d \geqslant \frac{1}{b} \sum_{i=2}^{b+1} L(i) - L(b+1). \tag{13}$$

Next, the upper bound is shown. In FIFO, let n_x and n denote the number of references to page x and the number of references to any page, respectively, during the stay of page x in the buffer. The probability of reference to x is given by

$$p(x) = n_x/n. \tag{14}$$

Since x is paged out after page faults have occurred b times, the fault rate is given by

$$f_{\mathrm{FIFO}} = b/n. \tag{15}$$

Considering the behavior of only x, the fault rate is also given by

$$f_{\mathrm{FIFO}} = 1/(n_x + 1). \tag{16}$$

Therefore, using (14), (15), and (16), the fault rate is given by

$$f_{\mathrm{FIFO}} = 1 - bp(x). \tag{17}$$

This implies that $p(x)$ is the average of the probability of reference to each page in the buffer.

In order to obtain the lower bound of $p(x)$, we consider how many steps x goes down in the LRU stack during its stay in the buffer. If all $b - 1$ pages which have stayed in the buffer at the time of loading x are referenced before their page-outs, x will go down $b - 1$ steps in the stack. Since a page fault occurs $b - 1$ times after that, x will go down $b - 1$ steps again. Therefore, the maximum number of steps is $2(b - 1)$ and $p(x)$ is equal to or greater than the average in the worst case shown as

$$p(x) \geqslant \frac{1}{2b-1} \sum_{i=1}^{2b-1} L(i). \tag{18}$$

The fault rate of LRU is given by

$$f_{\mathrm{LRU}} = 1 - \sum_{i=1}^{b} L(i). \tag{19}$$

Consequently, using (17), (18), and (19), the difference in fault rates between LRU and FIFO is given by

$$d \leqslant \sum_{i=1}^{b} L(i) - \frac{b}{2b-1} \sum_{i=1}^{2b-1} L(i). \tag{20}$$

Substituting (13) and (20) into (2), it is ascertained that the upper and lower bounds of the relative difference in performance for FIFO to LRU are monotone decreasing functions at low fault rate. The proof is presented in the Appendix. This conclusion is consistent with our experimental results shown in Fig. 4(a).

D. Performance of Simplified LRU's

LRU is suitable for data references with locality but full implementation of this algorithm is not practical because of stack manipulations. Simplified LRU's such as FINUFO and FIVE are, however, of practical use. In our experiments, although FIVE had more bits for an LRU-flag than FINUFO, FINUFO was better than FIVE. This is because the performance of simplified LRU's strongly depends on the period T for which information about recent references to pages is collected. The remainder of this section will discuss the importance of the reset timing of LRU-flags for simplified LRU's, by evaluating T for LRU, FINUFO, and FIVE.

The collection time T is simply defined as the number of faults during the reference strings which determines the current state of LRU-flags or an LRU stack, although it is more natural to define T as the number of references during that time. This definition of T does not lose generality because T is a function of the fault rate and the number of faults is equal to the product of the fault rate and the number of references.

In the case of LRU, the state of the LRU stack is determined by the reference strings including recent faults of b times (proof omitted here), that is,

$$T_{\mathrm{LRU}} = b. \tag{21}$$

In FINUFO, the state of the LRU-flags is determined by the reference strings during only one cycle of search for a reset flag as mentioned in Section III-A. The number of faults during one cycle is equal to the number of reset flags, that is,

$$T_{\mathrm{FINUFO}} = rb \tag{22}$$

where r denotes the ratio of reset flags in the buffer.

Since the number of flags turned off is equal to the number of flags turned on in the stationary state, the following equation is obtained:

$$\frac{b(1-r)}{br} f = r_0(1-f) \tag{23}$$

where f is the fault rate and r_0 denotes the probability that the flag of a referenced page in the buffer was off directly before referencing. The left-hand side is the product of the fault rate and the number of flags turned off per fault. The right-hand side is the number of flags turned on per reference.

Although r_0 cannot be arithmetically expressed, it must be lower than r because of locality. Therefore, if r approximates r_0, (23) is substituted by

$$\frac{b(1-r)}{br} f \leqslant r(1-f). \tag{24}$$

Consequently, using (21), (22), and (24),

$$T_{\mathrm{FINUFO}} \geqslant T_{\mathrm{LRU}} \frac{\sqrt{f(4-3f)} - f}{2(1-f)}. \tag{25}$$

Finally, the FIVE algorithm mentioned in Section III-A explicitly indicates

$$T_{\mathrm{FIVE}} = 5. \tag{26}$$

Every collection time is plotted in Fig. 6. T_{LRU} and T_{FINUFO} were evaluated by applying the parameter of ENV in (8) to (6) and (7) because they depend on the degree of locality. Furthermore, experimentally measured data for T_{FINUFO} is also shown to compensate the theoretical approximation of (24). The following two features are observed from this figure.

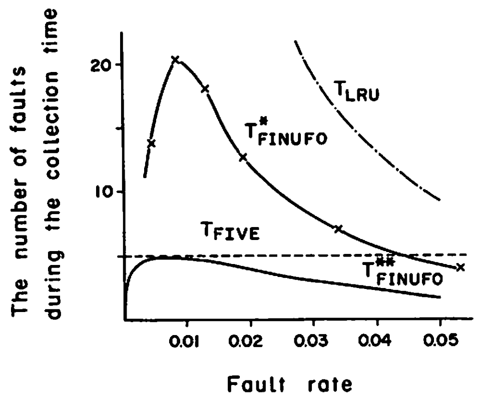

Fig. 6. The collection time of LRU-information for LRU and simplified LRU's.

1) The difference between T_{LRU} and the others increases at a low fault rate.

2) T_{FINUFO} tends to be longer than T_{FIVE} at a low fault rate.

These features are consistent with the experimental results shown in Fig. 4(b) and (c). This means that the performance of LRU and simplified LRU's strongly depends on T and that the reset timing of LRU-flags is very important for simplified LRU's.

V. Conclusions

The paging behavior of data referenced by a compiler was analyzed. This compiler translates a highly modularized program based on recent new programming methodologies such as stepwise refinement and data abstraction.

Experimental analyses have shown the following.

1) The optimum page size is rather small in comparison with that for procedures, that is, 256 bytes.

2) Locality is evident but is less than that for procedures.

3) Each replacement algorithm is efficient in the descending order of LRU, FINUFO, FIVE, and FIFO.

In order to confirm the experimental results, theoretical analyses have been performed especially with respect to the difference in performance between algorithms. Since only locality was assumed, the following results are not limited to compilation only.

1) The difference in performance between LRU and FIFO increases at a low fault rate.

2) The performance of simplified LRU's strongly depends on the reset timing of LRU-flags.

On the basis of these results, the replacement algorithm was changed from FIFO to FINUFO and the page size was also changed from 512 bytes to 256 bytes in the second version of the SPL compiler. Thus the fault rate was reduced by about 50 percent, that is, 20.2 percent by the former and 28.7 percent by the latter.

Appendix

This section proves that the upper bound function g_U and the lower bound function g_L mentioned in Section IV-C are monotone decreasing functions at low fault rate. For reasons of mathematical convenience, and with no loss of accuracy, the continuous function $l(x)$ is introduced as follows:

$$L(i) = \int_{i-1}^{i} l(x)\,dx, \quad i = 1, 2, \cdots. \tag{27}$$

From the feature of $L(i)$ expressed in (1), $l(x)$ is characterized by

$$0 < l(v) < \frac{1}{v-u}\int_u^v l(x)\,dx < l(u) \leq 1, \quad 0 \leq u < v. \tag{28}$$

Using (2), (19), and (20), g_U is given by

$$g_U(f) = \frac{\int_0^b l(x)\,dx - \frac{b}{2b-1}\int_0^{2b-1} l(x)\,dx}{1 - \int_0^b l(x)\,dx} \tag{29}$$

and its derivative is obtained as follows:

$$\left(1 - \int_0^b l(x)\,dx\right)^2 (dg_U/db) = l(b) - \frac{1 - \int_0^b l(x)\,dx}{2b-1} \cdot \left\{ bl(2b-1) - \frac{\int_0^{2b-1} l(x)\,dx}{2b-1} \right\} - \frac{b}{2b-1} l(b) \int_0^{2b-1} l(x)\,dx. \tag{30}$$

From (28),

$$\text{the right-hand side of (30)} > l(b) - \frac{bl(2b-1) - l(2b-1)}{2b-1} - \frac{bl(b)}{2b-1} = \frac{b-1}{2b-1}\{l(b) - l(2b-1)\} > 0. \tag{31}$$

Since $db/df < 0$ from (19), consequently,

$$dg_U/df = (dg_U/db)(db/df) < 0. \tag{32}$$

This means that g_U is a monotone decreasing function.

Next, using (2), (13), and (19), g_L is given by

$$g_L(f) = \frac{\frac{1}{b}\int_1^{b+1} l(x)\,dx - L(b+1)}{1 - \int_0^b l(x)\,dx} \tag{33}$$

and its derivative is obtained as follows:

$$\left(1 - \int_0^b l(x)\,dx\right)^2 (dg_L/db) = \left\{\frac{1}{b}\int_1^{b+1} l(x)\,dx - l(b+1)\right\}$$
$$\cdot \left\{l(b) - \frac{1}{b}\left(1 - \int_0^b l(x)\,dx\right)\right\} + \left\{l(b) - l(b+1)\right\}$$
$$\cdot \left\{1 - \int_0^b l(x)\,dx - l(b)\right\} + l(b)\{l(b) - L(b+1)\}. \tag{34}$$

From (27), the first factor of the first term, the first factor of the second term and the third term in the right-hand side of (34) are positive. Therefore, the right-hand side must be positive if the second factor of the first term and the second factor of the second term are positive. That is,

$$1 - \int_0^b l(x)\,dx > l(b) > \frac{1}{b}\left(1 - \int_0^b l(x)\,dx\right). \tag{35}$$

This sufficient condition is satisfied at $b > 30$, that is, at $f < 0.01$ in our experiments. Consequently, in this case,

$$dg_L/df = (dg_L/db)(db/df) < 0. \tag{36}$$

Thus, g_L is a monotone decreasing function at low fault rate.

Acknowledgment

The authors are indebted to I. Nakata of Tukuba University, Y. Yoshizawa and T. Nishigaki of the Systems Development Laboratory, Hitachi, Ltd., for invaluable technical assistance, especially with respect to the paging behavior of programs in virtual storage systems.

References

[1] E. W. Dijkstra, "Note on structured programming," in *Structured Programming*, O.-J. Dahl, E. W. Dijkstra, and C.A.R. Hoare, Eds. New York: Academic, 1972, pp. 1-82.
[2] N. Wirth, "Program development by stepwise refinement," *Commun. Ass. Comput. Mach.*, vol. 14, pp. 221-227, Apr. 1971.
[3] B. Liskov and A. Snyder, "Abstraction mechanism in CLU," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 564-576, Aug. 1977.
[4] T. Chusho and T. Hayashi, "Two-stage programming: Interactive optimization after structured programming," in *Proc. UJCC 1978*, pp. 171-175.
[5] E. G. Coffman and L. C. Varian, "Further experimental data on the behavior of programs in a paging environment," *Commun. Ass. Comput. Mach.*, vol. 11, pp. 471-474, July 1968.
[6] L. A. Belady and C. J. Kuehner, "Dynamic space-sharing in computer systems," *Commun. Ass. Comput. Mach.*, vol. 12, pp. 282-288, May 1969.
[7] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Syst. J.*, vol. 9, pp. 78-117, 1970.
[8] P. J. Denning, "Virtual memory," *ACM Comput. Surveys*, vol. 2, pp. 153-189, Sept. 1970.
[9] E. G. Coffman and P. J. Denning, *Operating System Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1973.
[10] E. Gelenbe, "A unified approach to the evaluation of a class of replacement algorithms," *IEEE Trans. Comput.*, vol. C-22, pp. 611-618, June 1973.
[11] S. A. Schuster, E. A. Ozkarahan, and K. C. Smith, "A virtual memory system for a relational associative processor," in *Proc. NCC 1976*, pp. 855-862.
[12] J. Rodrigues-Rosell, "Empirical data reference behavior in data base systems," *Computer*, vol. 9, pp. 9-13, Nov. 1976.
[13] S. W. Sherman and R. S. Brice, "Performance of a database manager in a virtual memory system," *ACM Trans. Database Syst.*, vol. 1, pp. 317-343, Dec. 1976.
[14] T. Hayashi *et al.*, "Top-down structured programming language for real-time computer systems–SPL," *Hitachi Rev.*, vol. 26, pp. 333-338, Oct. 1977.

Takeshi Chusho was born in Marugame, Japan, in 1946. He received the B.S. and M.S. degrees in electronic engineering from Tokyo University, Tokyo, Japan, in 1969 and 1971, respectively.

He is a Researcher at the Systems Development Laboratory, Hitachi, Ltd., Kawasaki, Japan. Since joining the company in 1971, he has worked on the design and development of languages, compilers, and other software tools. His current research interests include programming methodology, language, and compiler, and software testing.

Mr. Chusho is a member of the Association of Computing Machinery, SIGPLAN, the Institute of Electronics and Communication Engineers of Japan, and the Information Processing Society of Japan.

Toshihiro Hayashi was born in Osaka, Japan, in 1945. He received the B.S. degree in engineering (electronic communication) from Osaka University, Osaka, Japan, in 1967.

Since 1967 he has been with Hitachi, Ltd., Ibaraki, Japan, where he is currently a Senior Engineer. Since 1970 he has been in charge of development of various software engineering facilities such as programming languages and systems, testing languages and systems, and CAD systems for software production. He is currently interested in the application of requirement engineering and data base technology to industrial embedded computer systems.

Mr. Hayashi was an active member of International Purdue Workshop of Japan and is a member of Information Processing Society of Japan.