A Mathematical Approach to Program Behaviour And Look-Ahead Replacement Algorithms

BY

Otto Spaniol

A 74-16

Summary

A model for program behaviour is developed which is well suitable for higher level language programs.

On the basis of this model a demand prepaging rule and a look-ahead algorithm (LAA) is derived which turns out to be superior to the usual demand paging algorithms.

Zusammenfassung

Es wird ein Programmstrukturmodell beschrieben, welches in besonderem Maße Lokalitätseigenschaften von Programmen in höheren Programmiersprachen berücksichtigt.

Auf der Grundlage dieses Modells wird ein Look-ahead-Nachladealgorithmus entwickelt, der leistungsfähiger ist als die üblichen Demand-Paging-Algorithmen.

Contents

1. DESCRIPTION OF THE MODEL

1.1 Replacement algorithms

In this paper a two-level paged memory system ist studied. It will be presumed that only the first level (main memory MM) is directly addressable; moreover it is assumed that the second level (secondary memory SM) is large enough to keep all of the program and the data.

A replacement process is composed of two parts :

a) Loading a page $B_i \subset$ SM, which is not yet present in MM, into $X_{j(i)} \subset$ MM ($B_i \rightarrow X_{j(i)}$)
and

b) Reloading the old contents of $X_{j(i)}$ back into SM.

The reloading may be suppressed if there are still free pages in MM, or if the contents of $X_{j(i)}$ have not changed during the residency of $X_{j(i)}$ in MM (this may be recognized by a read-only bit).

A replacement algorithm is determined by the following attributes :

1. Look-ahead criterion : Determination of times at which replacements are made.
2. Loading problem : Which pages (and how many) should be loaded at a time ?
3. Replacement problem : If MM is full, which pages are to be replaced in favour of the incoming ones ?

Besides the choice of a special algorithm, the traffic between SM and MM is governed by the page size, the utilization of words in a page, the costs of replacing a group of pages, the style of programming etc. (cf[2]).

Since any page fault reduces the speed of computation (if SM is not directly addressable [14]), the choice of the replacement algorithm will be a very crucial problem.

Definition 1 :

a) *Let* $w = x_1\ x_2 \dots x_t \dots x_T$ *be the reference string of a program (computation), i.e. the sequence of addresses the program needs successively.*
(The reason for choosing the string of addresses instead of the string of pages referenced will become clear from the discussions in section 2 of the paper).

b) $RA^{[t]} := \{ B_i \mid X_{j[i]} \to B_i \text{ at time } t \}$
(set of pages to be brought in at time t by a replacement algorithm RA).

c) *The costs of loading a group of k pages are given by a mapping*
$h : N_0 \to R_+$
where $h(0) := 0$, $h(1) := 1$, $h(k) \geq h(k-1)$.
Except from these restrictions which should be satisfied for any interpretation of " costs ", mapping h may be arbitrary.

Interpretations of h(k) could be : time of loading a group of k pages or equipment costs or other meaningful criteria or combinations of such criteria.

In part c) of definition 1, only the loading costs are regarded. This simplification will be motivated by the fact, that in practice the replacement costs tend to be a fixed fraction of the loading costs [5]. An exact consideration of reloading costs results in an unsuitable complication.

Definition 2 :

$$C\ (\ RA,\ n,\ w\ ) := \sum_{t=1}^{T} h(|RA^{(t)}|)$$

(*total costs of processing a reference string w using a main memory consisting of n pages and RA as a replacement policy*).

The objective is to determine a replacement algorithm RA which is easy to implement and which results in minimal or near minimal costs for the class of reference strings which correspond to " meaningful " programs.

1.2 Classification of replacement algorithms

According to the criterion which causes the loading (and reloading, if MM is full) of one or more pages at a time the following classes of replacement algorithms will be distinguished :

Class	Loading condition	$\lvert RA^{(t)} \rvert$
Demand paging	$x_t \notin MM^{(t)}$ (page fault)	1
Demand prepaging	$x_t \notin MM^{(t)}$	≥ 1 (including the missing one)
Look-ahead	$x_t \notin MM^{(t)} \setminus D^{(t)}$	≥ 0

The subset $D^{(t)}$ of $MM^{(t)}$ characterizes the look-ahead behaviour of the algorithm.

The set of pages to be loaded in advance may be empty, even if the loading condition is satisfied (if there is no page fault, i.e. $x_t \in D^{(t)}$, and all pages which are candidates to be loaded are already present in $MM^{(t)}$).

1.2.1 Demand paging

Most of the papers are confined to demand paging replacement algorithms (DRA) only. For this class of algorithms there is no loading decision (pages are loaded at page faults only). Thus the structure of demand paging is much simpler than the behaviour of non-demand paging algorithms.

Furthermore, $h(|DRA^{(t)}|) = |DRA^{(t)}|$ and therefore :

$$C(DRA, n, w) = \sum_{t=1}^{T} |DRA^{(t)}| = \sum_{t=1}^{T} \varepsilon_t$$

where $\varepsilon_t = \begin{cases} 1 & \text{page fault at time } t \\ 0 & \text{no page fault} \end{cases}$

Hence the hitrate (probability of finding a data in MM) should be a better (but not necessarily optimal [4]) performance criterion for DRA than for non-DRA.

Another reason for the restriction on DRA is the following result [13]:

Theorem 1 :

If $h(k) \geq k$ $(k \geq 0)$ *then for any replacement algorithm RA there exists a demand paging algorithm DRA satisfying*

$C(DRA,n,w) \leq C(RA,n,w)$

for all n *and all* w.

Proof :

DRA defers any look ahead replacement made by RA until the time when the corresponding page is referenced.
If $h(k) \geq k$, the costs of DRA never exceed the costs of RA.
A formal proof is given in [13] and [5].

The condition $h(k) \geq k$ seems to be satisfied for some storage systems and some interpretations of the cost functions, whereas for other systems (rotating devices etc.) $h(k)=1+\varepsilon(k-1)<k$ $(k \geq 2)$ may be motivated.

Moreover, if the costs are interpretated as time of transport, it should be possible to get $h(k)<k$ $(k \geq 2)$ even for nonrotating devices by providing mechanisms which enable us to load in parallel. For example, if switching times are rising logarithmically and are negligible compared to transport times, $h(k)=1+\varepsilon \cdot \lceil ld(k) \rceil < k$ $(k \geq 2)$ may be motivated.

In these cases the study of properties of non-demand paging algorithms will be interesting.

The demand paging algorithms only differ in the replacement criterion (LRU, FIFO, RANDOM, A_0, B_0, CYCLE n,...).

There are interesting subclasses of demand paging, e.g. stack algorithms and priority algorithms [5] . It will turn out that for the look-ahead algorithm LAA, which is derived in this paper, the stack property is not fulfilled.

1.2.2 Demand prepaging and look-ahead algorithms

To the authors knowledge only two papers ([3] , [10]) deal with the construction of non-demand paging algorithms (if only the first storage level is directly addressable).

Baers version of OBL (One Block Look-ahead) and its modifications SL (Spatial Locality) and TL (Temporal Locality) are demand prepaging algorithmswhereas Josephs OBL and the extension SP (Simple Prediction) of OBL are look-ahead-algorithms.

In summary, Baers OBL works as follows : besides the missing page B_q (which is placed on the top of the LRU-Stack) its neighbour B_{q+1} (if not yet present in MM) is loaded (on the bottom place of the stack).

Thus B_{q+1} is replaced at the occurence of the next page fault if there was n° reference on this page in the mean time.

By this organization which is closely related to the LRU-principle, MM is not burdened with pages which have been prepaged superfluously, but on the other side the effect of

prepaging is greatly reduced.

Josephs version of OBL is a look-ahead-algorithm, because B_{q+2} is loaded in advance if a prepaged page B_{q+1} is referenced.

In the extended version SP of OBL, more than one position of MM is reserved to hold pages which are loaded in advance. It will turn out that both OBL and SP are special cases of LAA which will be constructed in section 3 by means of a mathematical model. It should be noted that OBL, SP, SL and TL have been derived mainly by heuristical arguments.

1.3 Locality, Models of program behaviour

A demand paging algorithm is optimal if it replaces the page of MM which has greatest forward distance by the page with smallest forward distance, i. e. by the missing page (algorithm B_0 of Belady [4]).

Correspondingly an optimal demand prepaging algorithm which is able to load k pages at the same time will replace the k pages of MM with greatest forward distance by the k pages which are not yet present in MM and which have smallest forward distances.

If the reference string is not known in advance we are forced to predict the future behaviour of the references by the recent past and to replace pages which we expect to have greatest forward distances. The same applies for look-ahead algorithms.

By doing so, it is assumed implicitly that any meaningful pro-

gram doesnot refer to its pages or its addresses at random. For most programs the following properties of locality are more or less satisfied [5] , [12]:

L 1 : *References are concentrated nonuniformly over the pages (or the addresses) of a program.*

L 2 : *Correlation of recent past and immediate future tends to be high, cf. the working set model* [5 - 7].

L 3 : *There are special reference patterns which are observed again and again. This is due to the looping structure of many programs, programing style and other reasons.*

There is a lot of other locality assumptions which are fitting to some extent for a given program. The following property which is very restrictive should be mentioned [5].

L 4 : *As a function of time, the frequency with which a page is referenced, is quasi-stationary.*

In the last few years many models of program behaviour have been derived which consider to some extent properties L1 - L4 (cf. [12])or properties of special storage systems [8].

Most of the models are probalistic; their agreemant with real programs has to be tested by simulations.

1.4 Description of another model of program behaviour.

Regions of a program

In the following a mathematical model is presented which introduces properties of locality which are observed for many parts of (FORTRAN-) programs. On the base of this model which may be used especially for looping programs, demand prepaging and look-ahead algorithms will he derived.

1.4.1 Regions

Programmers tend to concentrate data of the same " type " in contiguous storage areas (regions) B_1, B_2,... . This is due to programming habits and the structure of programming languages.

For example, we have a region I of the instructions of a program, a region V of a simple variables, regions A_1, A_2,... of arrays, a region C of constants (called " output catchment area" in [9, 11]) etc. .

Numerical results of Gibson [9] and Joseph [11] show that for many programs a partition of S" into 4 or 5 regions will be sufficient. To some extent, this holds indepedently of the page size.

The number of pages which a region occupies in "" varies with the time t. Analogously the number of pages in "" which are associated to a region should be influenced by the activity and the structure of the region concerned. In this section, however, the following simplifications are introduced :

1, To each region $R_i(t)$ of SM corresponds an area $X_i(t)$ of MM. No fragmentation problems are considered, i. e. both $R_i(t)$ and $X_i(t)$ consist of full pages.

2. Both R_i and X_i are assumed as independent of time t (for extensions of these assumption see 3.2.).

Thus we may restrict on one region R, one area X and a substring $w^R := x_1^R, \ldots, x_{T_R}^R$ of w (containing only the addresses of region R).

For denotational purposes, R is composed of pages $B_1, \ldots, B_a$ of size k :

$R := \bigcup_{i=1}^{a} B_i := \{1, \ldots, k \cdot a\}$ (a page will be identified with the set of addresses which are contained in it).

Moreover, X consists of $n<a$ pages of the same size.

$$X := \bigcup_{i=1}^{n} X_i$$

3. Finally let us suppose that X is filled with distinct pages of R_i

$X_i \subset R$ ($i=1,\ldots,n$), $X_i \neq X_j$ ($i \neq j$).

(There is neither placement nor replacement problem if there are still free places in MM - with the exception of filling X as quick as possible with pages of R).

1.4.2 Distribution of address jumps

Let x_{t-1}^R be the address of region R which has been referenced most recently.

In order to predict x_t^R from $x_{t-1}^R, x_{t-2}^R, \ldots$, we define a previous address z_t^R of x_t^R. The difference of these addresses

is a random variable whose properties are discussed in 1.4.3.

Definition 3 :

a) $z_t^R = z_t^R(x_t^R, h^R)$ *is called the previous address of* x_t^R

$\Longleftrightarrow$

1. $z_t^R = x_{t-i_o}^R \in \bigcup_{i=1}^{h^R} \{ x_{t-i}^R \}$

2. $|x_t^R - z_t^R| \leq |x_t^R - x_{t-i}^R|$ *for* $i=1,\ldots,h^R$

3. $|x_t^R - z_t^R| < |x_t^R - x_{t-i}^R|$ *for* $i=1,\ldots,i_o-1$

b) $y_{i,z}^R(t) := Pr\{ x_t^R - z_t^R = i \mid z_t^R = z \}$

Thus $y_z^R(t) := (y_{i,z}^R(t))_{i\in\mathbb{Z}}$ *is the distribution of address jumps* $x_t^R - z_t^R$ *in region R starting from a position z of the previous address.*

Now we are ready for our basic assumptions of locality:

L I* : *Address jumps are regarded as independent random variables. Moreover, it is assumed that there are relations*

$$y_{i,z}^R(t) \mathcal{R} y_{j,z}^R(t)$$

where $(i,j)\in J \subset \mathbb{Z}\times\mathbb{Z}$, $\mathcal{R} = \mathcal{R}(i,j,a,R) \in \{\leq, \geq, <, >, =, \neq\}$

and $\mathcal{R}$ *and* J *are independent of* t.

The structure of J *and the relations* $\mathcal{R}$ *will be discussed for several regions of importance.*

L 2* : *Address jumps are assumed as independent of the past with the exception of the position z of* z_t^R.
If $h^R=1$, *this is the usual Markow-condition.*

Remarks :

a) Since time-indepency of relations $\mathcal{R}$ is assumed, the time index of the distributions $y_{i,z}$ will be suppressed in the following.

b) Contrary to most of the models which have been presented by other authors (c.f. [12]) our model is free from restrictive stationary assumptions. The probabilities $y_{i,z}^R$ may change significantly in time without invaliditating the relations $\mathcal{R}$.

c) The structure of y_z^R depends on the window size h^R.

Obviously $|x_t^R - z_t^R|$ decreases if h^R increases.

If h^R is large, smaller address jumps will dominate over the larger ones.

d) A windowsize $h^R = o$ is of very limited interest since in this case any prediction basing on z_t^R is impossible.

1.4.3 Discussion of y_z^R for regions R of interest

1.4.3.1 Region I of instructions

In practice the sequence of instruction addresses consists of strings of purely sequential addresses connected by an address jump of length $\neq 1$.

Thus for $h^I = 1$ we have :

$$y_{1,z}^{I} >> y_{j,z}^{I} \quad (j \neq 1) \quad (z \neq k \cdot a)$$

or even

$$y_{1,z}^{I} > \sum_{j \neq 1} y_{j,z}^{I} \quad \text{(i.e. } y_{1,z}^{I} > 0.5\text{)}.$$

Other regions are governed by similar relations :
For example, if the difference of subsequent array elements is a constant, say m, in most cases (e.g. in matrix multiplication etc.) then

$$y_{m,z}^{A} >> y_{j,z}^{A} \text{ for } j \neq m \; ; \; z \begin{cases} \leq k \cdot a - m & m > 0 \\ \geq m & m < 0 \end{cases}$$

Very often, we have m=1 for the distance of two array elements which are referenced successively. In this case the distributions y_z^A and y_z^I are almost identical.

1.4.3.2 Region V of simple variables

Simple variable are rarely declared " at random ", i.e. there is a correlation between the use of two simple variable one after another and the distance of these variables in the declaration part of the program.

Therefore we may assume that smaller address jumps domineer over larger jumps in region V :

$$\left.\begin{array}{l} y_{i,z}^{V} \geq y_{i+1,z}^{V} \\ y_{-i,z}^{V} \geq y_{-i-1,z}^{V} \end{array}\right\} \quad i > 0 \quad (*)$$

If the page size k is sufficient (greater page sizes are smoothing many irregularities) the majority of relation (*)

should be satisfied for many programs even for the window-size $h^V = 1$ or $h^V = 2$. Similarly, relations $y^V_{i,z} \approx y^V_{-i,z}$ ($0 \leqslant i \leqslant \min (ak-z, z-1)$) may be motivated. A failure of these relations for some values of i is compensated in many cases (cf. section 2.3. of the paper).

There are other regions, e.g. the region C of constants (" output catchment area " [9]), for which the structure of the address jump distribution is similar to y^V_z.

2. THE HITRATE AS A PERFORMANCE CRITERION FOR LOOK-AHEAD ALGORITHMS

2.1 A formula for the hitrate Q

Let $x_t, z_t \in R = \bigcup_{i=1}^{a} B_i = \{ 1,\ldots,k\cdot a \}$.

$X = \bigcup_{i=1}^{n} X_i$, $X_i \neq X_j$ ($i \neq j$),

$X_i = B_{j(i)}$ ($i = 1,\ldots,n$; $j(i) \in \{ 1,\ldots, n \}$).

If the replacement algorithm with which the reference string w is processed is non-demand paging, the hitrate $Q^{(t)} := \Pr (x_t \in X)$ is not necessarily a good performance criterion since rising $Q^{(t)}$ (by prepaging additional pages at time t - 1) may result in high costs at time t - 1 as well as in a decline of the hitrate $Q^{(t+i)}$ ($i>o$) at later times. This well be seen from an example in section 2.2. Thus, increasing $Q^{(t)}$ might blow up the total costs of processing the reference string for several reasons.

Nevertheless the hitrate will be, in general, a good criterion if the number of pages which are prepaged at the same time is severely limited (by 2 or 3).

This restriction will be very important for the construction of our look-ahead algorithm LAA.

From our locality assumptions $L1^*$ and $L2^*$ we obtain for the hitrate $Q^{(t)}$:

$$Q^{(t)} = Pr\ (\ x_t \in X\ )$$

$$= \sum_{i=1}^{a} Pr\ (x_t \in B_i\ ,\ B_i \subset X\ )$$

$$= \sum_{i=1}^{a} \sum_{m=1}^{a} Pr\ (\ x_t \in B_i\ ,\ z_t \in B_m\ ,\ B_i \subset X\ )$$

$$= \sum_{i=1}^{a} \sum_{m=1}^{a} w_{m,i}^{(t)} \cdot Q_{i-m,m}^{(t)} \cdot r_m^{(t)}$$

where

$$w_{m,i}^{(t)} := Pr\ (\ B_i \subset X \mid x_t \in B_i\ ,\ z_t \in B_m\ )$$

$$Q_{i,s}^{(t)} := Pr\ (\ x_t \in B_{i+s} \mid z_t \in B_s\ )$$

$$r_m^{(t)} := Pr\ (\ z_t \in B_m\ )$$

Since z_t is not affected by the contents of X (cf. definition 3), $r_m^{(t)}$ and $Q_{i,s}^{(t)}$ are dependent of the reference string and the distribution of the address jumps but independent of the replacement algorithm.

$Q^{(t)}$ may be increased by restructuring methods [lo] (i. e. by changing y_z and $Q_{i,s}$) in order to guarantee that most of the address jumps remain in the same page, but in this paper only those regions are considered which are structured like regions I and V (cf. 1.4.3.).

It will be shown that for these regions $Q^{(t)}$ is increased if at time t-1 the neighbours B_{q-1} and B_{q+1} of the page B_q of x_{t-1} (which coincides with z_t if $h^R = 1$ is a possible choice) are loaded together with B_q.

2.2. Increasing $Q^{(t)}$ by loading the neighbours.

In the following the time indices of $Q_{i-m,m}$, $w_{m,i}$ and r_m are suppressed.

Let $z_t \in B_u$. Then we have the following result:

Theorem 2 :

$$\text{Let } Q^{(t)} := \sum_{m=1}^{a} r_m \cdot \sum_{i=1}^{a} w_{m,i} \cdot Q_{i-m,m}$$

$$\tilde{Q}^{(t)} := \sum_{m=1}^{a} r_m \cdot \sum_{i=1}^{a} \tilde{w}_{m,i} \cdot Q_{i-m,m}$$

a. Let

$$\tilde{w}_{m,u+j} := \begin{cases} w_{u,u+j} + d_1 & \text{if } m = u \,,\ j = c \\ w_{u,u+j} - d_2 & \text{if } m = u \,,\ j = c + sign(c) \\ w_{m,u+j} & \text{otherwise} \end{cases}$$

where:

$$c \in \mathbb{Z} \,,\ 0 \leq d_2 \leq d_1 \,,\ d_1 \leq 1 - w_{u,u+c} \,,$$

$$d_2 \leq w_{u,u+c+sign(c)} \,.$$

$$\text{If } Q_{i,u} \geq Q_{m,u} \quad \text{for } 0 \leq i < m \leq a \text{ and } 0 \leq -i < -m \leq a$$

then $\tilde{Q}^{(t)} \geq Q^{(t)}$.

b. Let

$$\tilde{w}_{m,u+j} := \begin{cases} w_{u,u+j}+d_1 & \text{if } m = u \,,\ j = c \\ w_{u,u+j}-d_2 & \text{if } \begin{cases} \text{either } m=u \,,\ j = c+sign(c) \\ \text{or } \quad m=u \,,\ j = -c-sign(c) \end{cases} \\ w_{m,u+j} & \text{otherwise} \end{cases}$$

If $0 \leq d_2 \leq d_1$ *and* $Q_{i,u} \geq Q_{m,u}$ *for* $0 \leq |i| < |m| \leq a$
then: $\tilde{\underline{Q}}(t) \geq \underline{Q}(t)$.

Remark: *The following rather unusual definition of sign (x) has been used:*

$$sign(c) := \begin{cases} +1 & \text{if } c > 0 \\ -1 & \text{if } c < 0 \\ 1 \text{ or } -1 & \text{if } c = 0 \end{cases}$$

Proof:

a. $\tilde{Q}(t) = Q(t) + r_u \cdot (Q_{c,u} \cdot d_1 - Q_{c+sign(c),u} \cdot d_2)$.

Thus $\tilde{Q}(t) \geq Q(t)$ if $Q_{c,u} \cdot d_1 \geq Q_{c+sign(c),u} \cdot d_2$.
This condition is satisfied by assumption.

b. is proved similarly.

As an immediate consequence of theorem 2 we obtain the following:

Theorem 3 :

Let $x_{t-1} \in B_q$.

If the assumptions of theorem 2 (relations between $Q_{i,u}$ *and* $Q_{m,u}$*) hold for a windowsize* $h^R = 1$ *(cf. section 2.3.1) then* $u = q$ *and* $Q(t)$ *is increased by loading at time t-1 the neighbours* $B_{q\pm1}$, $B_{q\pm2}$, ... *of the page* B_q *of* x_{t-1}.

If the relations in question hold for $h^R = h' > 1$ only then according to theorem 2 the page B_u of z_t and its neighbours $B_{u\pm1}, B_{u\pm2}, \ldots$ (if missing from MM) should be loaded at time t-1. But if $h' > 1$ then not necessarily $u = q$; the greater h', the less information about u will be available if the reference string is unknown in advance.

If $u \neq q$ then the loading of neighbours of B_q may be very inefficient. This will be seen immediately if one attempts to generalize theorem 3 on $h' > 1$.

But if the reference string is regular in some sense then even for larger windowsizes some results may be obtained concerning the loading problem. This will be seen from the following example:

Example :

Consider a region S in which small address jumps (and thus small page jumps, cf. 2.3.) domineer for windowsizes $h^S \geq 2$ but not for $h^S = 1$. This will be the case if a region is composed of two arrays whose elements are referenced alternatively (e. g. matrix multiplication).

Thus $z_t^S = x_{t-2}^S$ (in general).

Now let $x_{t-1}^S \in B_q \subset X^{(t)} \setminus D^{(t)}$ and $x_{t-2}^S \in B_r$ $(r \neq q)$.

According to theorem 2 the greatest value of $Q^{(t)}$ will be obtained if at time t-1 the page B_r of z_t^S and as many neighbours of B_r as possible are loaded into MM.

Thus $X^{(t)}$ consists of B_q, B_r and n-2 neighbours of $\underline{B_r}$. Analogously $Q^{(t+1)}$ is increased as much as possible if $X^{(t+1)}$ consists of B_r, B_q and n-2 neighbours of $\underline{B_q}$.

Hence if $Q^{(t)}$ is maximized, not only many replacements (which result in higher costs) are needed at time t-1 but also at time t (since maximizing $Q^{(t)}$ results in a decline of $Q^{(t+1)}$), at time t+1 and so on.

The best we can do in our example is to limit the number of pages which may be replaced at a time on $|X|/2$ or less, i.e. to partition S into $S = S_1 \cup S_2$ (corresponding to the two arrays which are contained in S) and X into $X = X_1 \cup X_2$ where $|X_1| = |X_2|$. Thus for the subregions S_1, S_2 we have that small address jumps domineer even for windowsizes $h^{S_i} = 1$.

The following remarkable conclusions should be drawn from this example:

a. The number N of pages which may be replaced at the same time has to be limited severely. Above all this will be necessary for regions which require a windowsize greater than 1 in order to guarantee the preponderance of small address jumps.

b. Moreover if N is small (compared to the size of the region) the hitrate at later times will (in general) not be reduced improperly by look-ahead replacements.

c. Finally a limitation of N (on N = 2 or N = 3) allows us to neglect the partition of SM and MM into regions and image areas without deterioring unduly the performance of the algorithm. Thus the very difficult problems of the determination of regions R_i in SM and of the adaptation of the image areas X_i in MM according to the changing activity of a region may be overcome.

These considerations will be very important for the construction of non-demand-paging algorithms in section 3 of the paper.

2.3. Discussion of the assumptions of theorem 2 for several regions of interest

Let $z := z_t^R \in \{1,\ldots,k\cdot a\}$ be the position of the previous address of x_t^R.

If $z \in B_s$ $(s \in \{1,\ldots,a\})$ then $z = (s-1)\cdot k+r$ $(0 \leqslant r < k)$

$$\stackrel{def}{=} a_r$$

Let $p_{r,s} := \dfrac{Pr(z = a_r)}{Pr(z \in B_s)}$

Thus $Q_{i,s} = Pr(x \in B_{i+s} \mid z = a_r)$

$$= \sum_{r=1}^{k} Pr(x \in B_{i+s} \mid z = a_r)\cdot p_{r,s}$$

$$= \sum_{r=1}^{k} Pr(i\cdot k-r+1 \leqslant x-z \leqslant (i+1)k-r \mid z=a_r)\cdot p_{r,s}$$

$$= \sum_{r=1}^{k} p_{r,s} \cdot \sum_{j=ik-r+1}^{(i+1)k-r} y_{j,a_r}$$

2.3.1. Regions structured like V

Theorem 4 :

a. If $y_{j,a_\kappa} \geq y_{j+sign(j)\cdot k,a_\kappa}$ $\quad [\, j \in [(i-1)k+1,(i+1)k-1] \,]$

then $Q_{i,s} \geq Q_{i+sign(i),s}$ $\quad (i \neq 0)$

b. Let $\upsilon \in \{-1,+1\}$.

If:

1. $y_{j,a_\kappa} = y_{-j,a_\kappa}$ for $\begin{cases} j=1,..,\kappa-1 & (\upsilon = +1) \\ j=1,..,k-\kappa & (\upsilon = -1) \end{cases}$

2. $y_{j,a_\kappa} \geq y_{j+sign(j)\cdot k,a_\kappa}$ for $\begin{cases} j=1,..,k-\kappa & (\upsilon = +1) \\ j=1,..,\kappa-1 & (\upsilon = -1) \end{cases}$

3. $T := \sum_{j=0}^{\min(\kappa-1,k-\kappa)} (y_{\upsilon\cdot j,a_\kappa} - y_{\upsilon\cdot(k-j),a_\kappa}) \geq 0$

then $Q_{0,s} \geq Q_{\upsilon,s}$.

Proof:

a. $$Q_{i,s} - Q_{i+sign(i),s} = \sum_{r=1}^{k} p_{r,s} \cdot \sum_{j=ik-r+1}^{(i+1)k-r} (y_{j,a_r} - y_{j+sign(j)\cdot k,a_r}) \geq 0.$$

b.

I. Let $\upsilon = +1$:

$$Q_{0,s} - Q_{1,s} = \sum_{r=1}^{k} p_{r,s} \cdot (d_1 + d_2 + d_3)$$

where $d_1 = \sum_{j=-r+1}^{-1} (y_{j,a_r} - y_{j+k,a_r})$

$d_2 = y_{o,a_r} - y_{k,a_r}$

$d_3 = \sum_{j=1}^{k-r} (y_{j,a_r} - y_{j+k,a_r}) \geq o$.

By assumption the following is obtained for d_1:

1. $r \leq k/2 + 1$:

$$d_1 = \sum_{j=1}^{r-1} (y_{j,a_r} - y_{k-j,a_r}) = \sum_{j=1}^{\min(r-1,k-r)} (y_{j,a_r} - y_{k-j,a_r})$$

2. $r > k/2 + 1$:

$$d_1 = \sum_{j=1}^{k-1} (y_{j,a_r} - y_{k-j,a_r}) - \sum_{j=r}^{k-1} (y_{j,a_r} - y_{k-j,a_r})$$

$$= o + \sum_{j=1}^{k-r} (y_{j,a_r} - y_{k-j,a_r})$$

$$= \sum_{j=1}^{\min(k-r,r-1)} (y_{j,a_r} - y_{k-j,a_r})$$

Hence: $d_1 + d_2 = T \geq o$ by assumption

and therefore $Q_{o,s} - Q_{1,s} \geq o$.

II. Let $v = -1$:

$$Q_{o,s} - Q_{-1,s} = \sum_{r=1}^{k} p_{r,s} \cdot (e_1 + e_2 + e_3)$$

where $e_1 = \sum_{j=-r+1}^{-1} (y_{j,a_r} - y_{j-k,a_r}) \geq o$

$e_2 = y_{o,a_r} - y_{-k,a_r}$

$e_3 = \sum_{j=1}^{k-r} (y_{j,a_r} - y_{j-k,a_r})$

1. $r > k/2 + 1$:

$$e_3 = \sum_{j=1}^{\min(k-r,r-1)} (y_{-j,a_r} - y_{-k+j,a_r})$$

2. $r \leqslant k/2 + 1$:

$$e_3 = \sum_{j=1}^{k-1} (y_{-j,a_r} - y_{-k+j,a_r}) - \sum_{j=k-r+1}^{k-1} (y_{-j,a_r} - y_{-k+j,a_r})$$

$$= \sum_{j=1}^{\min(k-r,r-1)} (y_{-j,a_r} - y_{-k+j,a_r})$$

Thus $e_2 + e_3 = T \geqslant o$ concluding the proof.

Remarks:

a. Except from the assumption $T \geqslant o$ all the conditions of theorem 4 are consequences of the relations

$$y_{i,a_r} \geqslant y_{i+sign(i),a_r} \qquad (i \neq o)$$

$$y_{i,a_r} = y_{-i,a_r} \qquad (o \leqslant i \leqslant \min(a \cdot k - a_r, a_r - 1))$$

which have been motivated in 1.4.3.2. for region V.

b. If $y_{i,a_r} \geqslant y_{i+sign(i),a_r}$ for $i \in [-k+1, k-1]$ then only the term d_2 (or e_2) of T might be negative. In all cases of practical interest this term should be compensated by the other terms. Moreover, these considerations indicate that our assumptions are satisfied rather for larger k than for small values of the page size.

c. The validity of the relation $Q_{o,s} > Q_{1,s}$ even for small windowsizes is supported by simulation results of Joseph [11].

d. As a result of theorem 4 in addition to page B_q of x_{t-1} (if missing from MM) its neighbours B_{q-1} and B_{q+1} should be loaded at time t-1.
This is supported by the fact that stronger assumptions are needed to establish $Q_{o,s} \geq Q_{\pm 1,s}$ than to derivate $Q_{\pm 1,s} \geq Q_{\pm 2,s}$ etc. .

2.3.2. Regions structured like I

Theorem 5 :

a. $Q_{o,s} > Q_{i,s}$ $(i \neq o , i \neq 1)$ *holds if the following assumptions are satisfied:*

(1) $y_{1,z} > \sum_{j \neq 1} y_{j,z}$ $(z \neq k \cdot a)$

(2) $p_{1,s} \cdot p_{k,s}$

(3) $$\sum_{\substack{j=o \\ j \neq 1}}^{k-1} (y_{j,a_1} + y_{-j,a_k}) + y_{-1,a_k} + \sum_{\substack{j \neq 1 \\ j \notin [(i-1)k+1,\,(i+1)k-1]}} y_{j,a_1} \geq y_{ik,a_1}$$

(4) $y_{j,a_1} \cdot y_{j,a_k}$ *for* $j \in [(i-1)k+1,\ ik]$

b. Let $y_{j,z} = y_{m,z}$ for $j,m \in \{-a+1,\ldots,a\} \setminus \{0,1\}$

I. If $y_{1,z} > y_{j,z}$ $\quad (z \neq a \cdot k\,,\ j \neq 0,\ j \neq 1)$

then $\underline{Q_{1,s} > Q_{i,s}}$ for $(s \neq a) \vee [i \notin \{-a+1,\ldots,-1\}]$

II. If

$$\sum_{r=1}^{k} p_{r,s} \cdot (y_{0,a_r} + y_{1,a_r} - y_{k,a_r} - y_{k+1,a_r}) + p_{k,s} \cdot (y_{-k+1,a_k} + y_{k+1,a_k}) \geq 2 \cdot p_{k,s} \cdot y_{1,a_k}$$

then $\underline{Q_{0,s} \geq Q_{1,s}}$.

Proof:

a. $Q_{0,s} - Q_{1,s}$

$$= \sum_{r=1}^{k} p_{r,s} \cdot \Big(\sum_{j=-r+1}^{k-r} y_{j,a_r} - \sum_{j=ik-r+1}^{(i+1)k-r} y_{j,a_r} \Big)$$

$$> \sum_{r=2}^{k-1} p_{r,s} \cdot \Big(\sum_{\substack{j=-r+1 \\ j \neq 1}}^{k-r} y_{j,a_r} + \sum_{j \neq 1} y_{j,a_r} - \sum_{j=ik-r+1}^{(i+1)k-r} y_{j,a_r} \Big)$$

$$+ p_{1,s} \cdot \Big(\sum_{j=0}^{k-1} y_{j,a_1} - \sum_{j=ik}^{(i+1)k-1} y_{j,a_1} \Big)$$

$$+ p_{k,s} \cdot \Big(\sum_{j=-k+1}^{0} y_{j,a_k} - \sum_{j=(i-1)k+1}^{ik} y_{j,a_k} \Big)$$

$$\geq p_{1,s} \cdot \Big(\sum_{\substack{j=0 \\ j \neq 1}}^{k-1} y_{j,a_1} + y_{1,a_1} - \sum_{j=ik}^{(i+1)k-1} y_{j,a_1} \Big)$$

$$+ p_{k,s} \cdot \Big(\sum_{j=-k+1}^{0} y_{j,a_k} - \sum_{j=(i-1)k+1}^{ik} y_{j,a_k} \Big)$$

$$> p_{1,s}\cdot\Big(\sum_{\substack{j=0\\ j\neq 1}}^{k-1} y_{j,a_1} + \sum_{j\neq 1} y_{j,a_1} - \sum_{j=ik}^{(i+1)k-1} y_{j,a_1}$$

$$+ \sum_{j=-k+1}^{0} y_{j,a_k} - \sum_{j=(i-1)k+1}^{ik} y_{j,a_k}\Big)$$

$$= p_{1,s}\cdot\Big(\sum_{\substack{j=0\\ j\neq 1}}^{k-1} (y_{j,a_1}+y_{-j,a_k}) + y_{-1,a_k} - y_{ik,a_1}$$

$$+ \sum_{j\in[(i-1)k+1,(i+1)k-1]\atop j\neq 1} y_{j,a_1}\Big)$$

$$\geq 0 .$$

b. If $i \neq 0$ then

$$Q_{1,s} - Q_{i,s}$$

$$= \sum_{r=1}^{k} p_{r,s}\cdot\Big(\sum_{j=k-r+1}^{2k-r} y_{j,a_r} - \sum_{j=ik-r+1}^{(i+1)k-r} y_{j,a_r}\Big)$$

$$= p_{k,s}\cdot\big(y_{1,a_k} - y_{(i-1)k+1,a_k}\big) .$$

If $s = a$ then $y_{1,a_k} = 0$ and $Q_{1,a} \geq Q_{i,a}$ fails if $y_{(i-1)k+1,a_k} > 0$, i. e. for $i \in \{-a+1,..,-1\}$.

If $s \neq a$ then by assumption $y_{1,a_k} > y_{(i-1)k+1,a_k}$ whence $Q_{1,s} > Q_{i,s}$ for all i.

II. $Q_{0,s} - Q_{1,s}$

$$= \sum_{r=1}^{k} p_{r,s}\cdot\Big(\sum_{j=-r+1}^{k-r} y_{j,a_r} - \sum_{j=k-r+1}^{2k-1} y_{j,a_r}\Big)$$

$$= \sum_{r=1}^{k-1} p_{r,s} \cdot (y_{o,a_r} + y_{1,a_r} - y_{k,a_r} - y_{k+1,a_r})$$

$$+ p_{k,s} \cdot (y_{o,a_k} - y_{k,a_k} + y_{-k+1,a_k} - y_{1,a_k})$$

$\geq$ o (by assumption).

Remarks:

a. Since $\sum_{r=1}^{k} p_{r,s} = 1$, $Q_{o,s} > Q_{i,s}$ $(i \neq o , i \neq 1)$ holds even if some of the probabilities $p_{r,s}$ are zero.

b. Theorem 5 shows that besides the "actual" page B_q of x_{t-1} its neighbour B_{q+1} has the highest priority of being loaded in advance (in comparison with all the other pages).

c. Similar results are obtained for the region A of an array satisfying: $y^A_{m,z} >> y^A_{j,z}$ for $j \neq m$.

If the page size k is notless $|m|$ (this should include most cases of interest) then in addition to B_q the page $B_{q+sign(m)}$ should be loaded in advance.

If $k < |m|$ the prepaging of $B_{q+sign(m)\cdot\lfloor m/k + 1 \rfloor}$ may be motivated.

d. Most of the assumptions of theorem 5 are conclusions of the discussions made in 1.4.3.1.; they should be satisfied in any case of practical interest. The conditions may be weakened in many cases but none of them may be omitted without more ado; this will be seen from the following rather artificial example:

Let

$$y_{j,a_r} = \begin{cases} 0.5 + d/2 & \text{if } j=1 \\ 0.5 - d/2 & \text{if } j=i\cdot k \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } r = 1,\ldots,k$$

$$p_{1,s} = p_{k,s} \neq 0 \qquad (\, d > 0 \,,\; s \neq a \,)$$

Thus only assumption (3) of theorem 5 is violated. We obtain:

$$\begin{aligned} Q_{0,s} - Q_{1,s} &= \sum_{r=1}^{k-1} p_{r,s}\cdot y_{1,a_1} - \sum_{r=1}^{k} p_{r,s}\cdot y_{ik,a_1} \\ &= (1-p_{k,s})\cdot(y_{1,a_1}-y_{ik,a_1}) - p_{r,s}\cdot y_{ik,a_1} \\ &= (1 - p_{k,s})\cdot y_{1,a_1} - y_{ik,a_1} \\ &= d - p_{k,s}\cdot(1+d)/2 \\ &< 0 \quad \text{if} \quad p_{k,s} > 2d/(1+d) \,. \end{aligned}$$

Thus if $p_{k,s} \neq 0$ then there exists $d > 0$ with $Q_{0,s} < Q_{i,s}$.

3. CONSTRUCTION OF REPLACEMENT ALGORITHMS

3.1. A prepaging rule PN applicable to all regions of interest

Summarizing the results of section 2 the following pages should be loaded at time t-1 in addition to page B_q of x_{t-1}:

a. B_{q-1} , B_{q+1} (region V)

b. B_{q+1} (region I)

c. B_{q+1} (region A, $y^A_{m,z} >> y^A_{j,z}$ $(j \neq m)$, pagesize $k < m$)

d. B_{q-1} (" A " " $k < -m$)

The subcases b. - d. are contained in the loading rule "prepaging of the neighbours (PN)" which has been proposed for region V.
Furthermore, considering region I for example, we remark that B_{q-1} will be present in most cases when B_q is referenced; thus the loading rules for I and V coincide very often.

Solutions of the replacement problem and the look-ahead problem (i. e. the determination of a subset $D^{(t)}$ of $X^{(t)}$ characterizing look-ahead replacements) which are basing on PN as a loading rule will be studied in the following sections of the paper.

3.2. The demand prepaging algorithm DPA

It is easy to see that the demand prepaging algorithm DPA which is governed by PN as a loading rule and LRU as a replacement policy is an extension of Baers version of OBL (One Block Lookahead, cf. [3]).
If $A^{(t)} := [a_1^{(t)}, \ldots, a_n^{(t)}]$ are the contents of the LRU-stack of the image area X of region R (where $a_i^{(t)}$ are the numbers of the pages which are present in X at time t) then DPA works as follows:

I. $x_t \in B_q$ where $q = a_m^{(t)}$ $(1 \leq m \leq n)$:

$$A^{(t+1)} := [q, a_1^{(t)}, \ldots, a_{m-1}^{(t)}, a_{m+1}^{(t)}, \ldots, a_n^{(t)}]$$

II. $x_t \in B_q$, $q \notin A^{(t)}$:

$$A^{(t+1)} := \begin{cases} [q, a_1^{(t)}, \ldots a_{n-3}^{(t)}, q+1, q-1] & \text{if (1)} \\ [q, a_1^{(t)}, \ldots a_{n-2}^{(t)}, a_{n-1}^{(t)}] & \text{if (2)} \\ [q, a_1^{(t)}, \ldots, a_{n-2}^{(t)}, a_n^{(t)}] & \text{if (3)} \\ [q, a_1^{(t)}, \ldots, a_{n-3}^{(t)}, a_{n-1}^{(t)}, a_n^{(t)}] & \text{if (4)} \\ [q, a_1^{(t)}, \ldots, a_{n-2}^{(t)}, q+v] & \text{if (5)} \\ [q, a_1^{(t)}, \ldots, a_{n-3}^{(t)}, a_n^{(t)}, q+v] & \text{if (6)} \\ [q, a_1^{(t)}, \ldots, a_{n-3}^{(t)}, a_{n-1}^{(t)}, q+v] & \text{if (7)} \end{cases}$$

(1) : $q-1, q+1 \notin A^{(t)}$

(the order of the neighbour pages is motivated by the fact that for most regions of interest $Q_{1,s} \geq Q_{-1,s}$ will be satisfied)

$$(2): \quad q-1 \,,\ q+1 \in A^{(t)} \setminus \{a_n^{(t)}\}$$

$$(3): \quad q-v = a_n^{(t)} \,,\ q+v \in A^{(t)} \setminus \{a_{n-1}^{(t)}\}$$

$$(4): \quad q-v = a_n^{(t)} \,,\ q+v = a_{n-1}^{(t)}$$

$$(5): \quad q+v \notin A^{(t)} \,,\ q-v \in A^{(t)} \setminus \{a_{n-1}^{(t)}, a_n^{(t)}\}$$

$$(6): \quad q+v \notin A^{(t)} \,,\ q-v = a_n^{(t)}$$

$$(7): \quad q+v \notin A^{(t)} \,,\ q-v = a_{n-1}^{(t)} \qquad (\ v \in \{-1,+1\}\ )\ .$$

The organization of an ensemble of regions $\tilde{R}_1, \ldots, \tilde{R}_m$ and of image areas $\tilde{X}_1, \ldots, \tilde{X}_m$ (where each pair $(\tilde{R}_i, \tilde{X}_i)$ is governed "locally" by DRA) is simplified considerably by extending DRA on $\tilde{R} := \bigcup_{i=1}^{m} \tilde{R}_i$ and $\tilde{X} := \bigcup_{i=1}^{m} \tilde{X}_i$ ("global" DRA).

This modification doesnot present any problem for $\tilde{R}$, since each $\tilde{R}_i$ is governed by the same loading rule; this is independent of the region concerned.

If $\tilde{X}$ is to be organized globally in agreement with LRU, then instead of 1 or 2 positions for each of the $\tilde{X}_i$ we should reserve α positions ($m \leq \alpha \leq 2m$) of $\tilde{X}$ to keep the pages which are loaded in advance from the subregions $\tilde{X}_1, \ldots, \tilde{X}_m$. Thus $\tilde{X}$ splits into a LRU-part $A^{(t)}$ (containing the pages which have been referenced in the recent past) and a "look-ahead"-part $B^{(t)}$.

If we confine us to a system which allows the existence of at most m image areas at the same time then $\alpha = |B^{(t)}| = m$ might be a rather good choice because in most cases only one page is to be prepaged at a time.

Thus if no multiprogramming is allowed and if a program may be partitioned into 4 or 5 regions of activity [9,11] a look-ahead-part consisting of $\alpha = 4$ pages should be sufficient.

But even in a multiprogramming environment it will be expected that it is possible to work efficiently with $|B^{(t)}| = 8$ in most cases.

The organization of $A^{(t)}$ and $B^{(t)}$ will be discussed in detail for the look-ahead-algorithm LAA which is presented in the next section.

3.3. The look-ahead algorithm LAA

As proposed in 3.2., $MM^{(t)}$ splits into a LRU-part $A^{(t)}$ and into a look-ahead part $B^{(t)}$.
For convenience, $D^{(t)} := B^{(t)}$ (i. e. look-ahead replacements will be made only if the data referenced is present in the look-ahead part of $MM^{(t)}$).
Moreover it will be assumed that $|B^{(t)}| = |B| = \text{const}$.
Thus, if the size of $MM^{(t)}$ is a constant the size of $A^{(t)}$ doesnot vary in time.

$A^{(t)} := [a_1^{(t)},\ldots,a_{|A|}^{(t)}]$, $B^{(t)} := [b_1^{(t)},\ldots,b_{|B|}^{(t)}]$
where a_i , $b_i \in \{1,\ldots,a\}$.

Furthermore, $B^{(t)}$ will be governed by the FIFO-principle since $B^{(t)}$ only contains pages which have not been referenced in the recent past.

Clearly there are other possibilities of organizing $A^{(t)}$ and $B^{(t)}$. Some minor problems will not be regarded in the following, e. g. formulas for $|A| < 3$, the loading problem for

marginal pages of MM etc. ; it will be obvious to complete the organization.

To shorten the notation the following abbreviations will be used:

Definition 4 :

Let r_t *be the number of the page which is referenced at time* t.

$r_t^+ := r_t+1$, $r_t^- := r_t-1$, $R_t := \{r_t^+ , r_t^-\}$.

By $e_A^{(t)}$ *we denote the page of* $A^{(t)}$ *which will be replaced by* r_t.

3.3.1. $r_t = a_m^{(t)} \in A^{(t)}$

$$A^{(t+1)} := [r_t, a_1^{(t)}, \ldots, a_{m-1}^{(t)}, a_{m+1}^{(t)}, \ldots, a_{|A|}^{(t)}]$$

$$B^{(t+1)} := B^{(t)}$$

3.3.2. $r_t \notin A^{(t)}$

According to the LRU-principle we have $a_1^{(t+1)} := r_t$.
The neighbours of r_t are loaded into $B^{(t+1)}$ if they are not yet present in $A^{(t)} \cup B^{(t)}$. If $r_t \in B^{(t)}$ this will be a real look-ahead step which may be done without interrupting the program. Thus if costs are given by "time" the definition of the costs of a replacement algorithm (cf. definition 1) should be modified in such a way that only non-look-ahead replacement costs are regarded. This presents an interesting argument in favour of non-demand-paging algorithms.

The top of $B^{(t+1)}$ consists of the page $e_A^{(t)}$ which is replaced from $A^{(t)}$ (if $e_A^{(t)} \in R_t$) and of the incoming neighbour(s) of r_t whereas the bottom pages of $B^{(t)}$ are replaced.

3.3.2.1. Organization of $A^{(t+1)}$

Two possible organizations are discussed in the following:

LRU 1 (replacement of $e_A^{(t)} = a_{|A|}^{(t)}$):

$$A^{(t+1)} := [r_t, a_1^{(t)}, \ldots, a_{|A|-1}^{(t)}].$$

If $e_A^{(t)} \in R_t$, then $b_1^{(t+1)} := a_{|A|}^{(t)}$ (cf. 3.3.2.2.).

LRU 2 (replacement of the page of $A^{(t)}$ which is not a member of R_t and which has been least recently used):

$$A^{(t+1)} := \begin{cases} [r_t, a_1^{(t)}, \ldots, a_{|A|-1}^{(t)}] & \text{if } a_A^{(t)} \notin R_t \\ [r_t, a_1^{(t)}, \ldots, a_{|A|-2}^{(t)}, a_{|A|}^{(t)}] & \text{if } a_{|A|}^{(t)} \notin R_t \cdot a_{|A|-1} \in R_t \\ [r_t, \ldots, a_{|A|-3}^{(t)}, a_{|A|-1}^{(t)}, a_{|A|}^{(t)}] & \text{otherwise} \end{cases}$$

Thus $e_A^{(t)} \notin R_t$ if $A^{(t)}$ is organized with LRU 2.

3.3.2.2. Organization of $B^{(t+1)}$

The organization of $B^{(t+1)}$ depends on whether $r_t \in B^{(t)}$ or $r_t \notin B^{(t)}$ as well as on whether $e_A^{(t)} \in R_t$ (which is impossible if $A^{(t)}$ is governed by LRU 2) or $e_A^{(t)} \notin R_t$.

3.3.2.2.1. $r_t = b_m^{(t)} \in B^{(t)}$, $e_A^{(t)} \notin R_t$

$$B^{(t+1)} := \begin{cases} [r_t^+, r_t^-, b_1^{(t)}, \ldots, b_{m-1}^{(t)}, b_{m+1}^{(t)}, \ldots, b_{|B|-1}^{(t)}] & \text{if } r_t^+, r_t^- \notin MM^{(t)} \\ [N(r_t), b_1^{(t)}, \ldots, b_{m-1}^{(t)}, b_{m+1}^{(t)}, \ldots, b_{|B|}^{(t)}] & \text{otherwise} \end{cases}$$

where

$$N(r_t) := \begin{cases} r_t^+ & \text{if } r_t^+ \notin MM^{(t)} \\ r_t^- & \text{if } r_t^+ \in MM^{(t)} \text{ , } r_t^- \notin MM^{(t)} \\ e_A^{(t)} & \text{if } r_t^+ \text{ , } r_t^- \in MM^{(t)} \end{cases}$$

3.3.2.2.2. $r_t = b_m^{(t)} \in B^{(t)}$, $e_A^{(t)} \in R_t$

Let $e_A^{(t)} = r_t^*$ ($* \in \{+,-\}$).

Using the abbreviation:

$$\bar{r_t^*} := \begin{cases} r_t^- & \text{if } r_t^* = r_t^+ \\ r_t^+ & \text{if } r_t^* = r_t^- \end{cases}$$

we obtain:

$$B^{(t+1)} := \begin{cases} [r_t^*, b_1^{(t)}, \ldots, b_{m-1}^{(t)}, b_{m+1}^{(t)}, \ldots, b_{|B|}^{(t)}] & \text{if } \bar{r_t^*} \in MM^{(t)} \\ [r_t^*, \bar{r_t^*}, b_1^{(t)}, \ldots, b_{m-1}^{(t)}, b_{m+1}^{(t)}, \ldots, b_{|B|-1}^{(t)}] & \text{if } \bar{r_t^*} \notin MM^{(t)} \end{cases}$$

3.3.2.2.3. $r_t \notin MM^{(t)}$, $e_A^{(t)} \notin R_t$

The neighbours of r_t are loaded (if missing from MM).

It has to be guaranteed that loading r_t^* doesnot replace $r_t^{\bar{*}}$ (if $r_t^{\bar{*}} = b_B^{(t)}$).

$$B^{(t+1)} := \begin{cases} B^{(t)} & \text{if (1)} \\ [r_t^+, r_t^-, b_1^{(t)}, \ldots, b_{|B|-2}^{(t)}] & \text{if (2)} \\ [r_t^*, b_1^{(t)}, \ldots, b_{|B|-1}^{(t)}] & \text{if (3)} \\ [r_t^*, b_{|B|}^{(t)}, b_1^{(t)}, \ldots, b_{|B|-2}^{(t)}] & \text{if (4)} \\ [r_t^*, b_1^{(t)}, \ldots, b_{|B|-2}^{(t)}, b_{|B|}^{(t)}] & \text{if (5)} \end{cases}$$

(1) : $r_t^+ , r_t^- \in MM^{(t)}$

(2) : $r_t^+ , r_t^- \notin MM^{(t)}$

(3) : $r_t^* \notin MM^{(t)}$, $r_t^{\bar{*}} \in MM^{(t)} \setminus \{b_{|B|}^{(t)}\}$

(4) : $r_t^* \notin MM^{(t)}$, $r_t^{\bar{*}} = b_{|B|}^{(t)}$, $A^{(t)}$ organized with LRU 1

(5) : $r_t^* \notin MM^{(t)}$, $r_t^{\bar{*}} = b_{|B|}^{(t)}$, $A^{(t)}$ organized with LRU 2

3.3.2.2.4. $r_t \in MM^{(t)}$, $e_A^{(t)} \in R_t$

Let $e_A^{(t)} = r_t^*$ ($* \in \{+,-\}$)

$$B^{(t+1)} := \begin{cases} [r_t^*, r_t^{\bar{*}}, b_1^{(t)}, \ldots, b_{|B|-2}^{(t)}] & \text{if (1')} \\ [r_t^*, b_1^{(t)}, \ldots, b_{|B|-1}^{(t)}] & \text{if (2')} \\ [r_t^*, b_{|B|}^{(t)}, b_1^{(t)}, \ldots, b_{|B|-2}^{(t)}] & \text{if (3')} \\ [r_t^*, b_1^{(t)}, \ldots, b_{|B|-2}^{(t)}, b_{|B|}^{(t)}] & \text{if (4')} \end{cases}$$

(1*) : $\bar{r}_t^* \notin MM^{(t)}$

(2*) : $\bar{r}_t^* \in MM^{(t)} \setminus \{b_{|B|}^{(t)}\}$

(3*) : $\bar{r}_t^* = b_{|B|}^{(t)}$, $A^{(t)}$ organized with LRU 1

(4*) : $\bar{r}_t^* = b_{|B|}^{(t)}$, $A^{(t)}$ organized with LRU 2

3.3.3. Summary, Implemantation of LAA

By the look-ahead algorithm LAA u pages ($0 \leq u \leq 3$) are loaded simultaneously if $x_t \in r_t \notin MM^{(t)} \setminus B^{(t)}$.
To simplify the organization some of the replacement rules may be omitted without reducing significantly the performance of the algorithm.
A version of LAA (governed by LRU 1 and disregarding unusual replacements) has been simulated on the CD 3300 and TR 440 computers of "Universität des Saarlandes, Saarbrücken (W-Germany)". Simulation results are given in the appendix.

3.4. Properties of LAA

3.4.1. Extension of OBL and SP

Any complete implementation of LAA is an extension of OBL [3,11] and SP [11].
For both OPL and SP the number of pages to be brought in at the same time is restricted by 2 (whereas up to 3 pages are loaded simultaneously by LAA).

Each of the algorithms OBL, SP and LAA is governed by the PN-principle (prepaging of neighbours); this differs from the modifications SL (spatial locality) and TL (temporal locality) of OBL which have been discussed in [3].
Furthermore

$$|B^{(t)}| = \begin{cases} 1 & \text{OBL} \\ m & \text{SP} \\ \alpha \quad (m \leq \alpha \leq 2m) & \text{LAA} \end{cases}$$

where m denotes the number of image areas which are simultaneously present in MM.
Simulation results of Joseph are obtained for $|B^{(t)}| = 4$ (considering the fact that for many programs four areas of activity are sufficient).

3.4.2. Non-stack-property of LAA

Since $B^{(t)}$ is governed by FIFO (which is not a stack algorithm) and $|A^{(t)}|$ and $|B^{(t)}|$ have been assumed to be time-invariant, LAA is not a stack algorithm.
To show this explicitly we must construct a reference string (of page numbers) $\omega = r_1 r_2 \ldots r_T$, a number $i \in \mathbb{N}$ and a time t with:

$$B^{(t)}(i,\omega) \not\subseteq B^{(t)}(i+1,\omega)$$

where $B^{(t)}(i,\omega)$ are the contents of the look-ahead-part B of MM at time t, if $|B| = i$ and the reference string ω is processed.

Example 1 (Organization of $A^{(t)}$ with LRU 1):

Let $|A^{(t)}| = 2$.

t	r_t	$A^{(t)}$	$B^{(t)}_{(4,w)}$	$B^{(t)}_{(5,w)}$
1	2			
2	6	2	3,1	3,1
3	1o	6,2	7,5,3,1	7,5,3,1
4	4	1o,6	11,9,7,5	11,9,7,5,3
5	16	4,1o	3,5,11,9	11,9,7,5,3
6		16,4	17,15,3,5	17,15,11,9,7

Example 2 (Organization of $A^{(t)}$ with LRU 2):

Let $|A^{(t)}| = 2$.

t	r_t	$A^{(t)}$	$B^{(t)}_{(4,w)}$	$B^{(t)}_{(5,w)}$
1	2			
2	5	2	3,1	3,1
3	8	5,2	6,4,3,1	6,4,3,1
4	2	8,5	9,7,6,4	9,7,6,4,3
5	11	2,8	3,1,9,7	1,9,7,6,3
6		11,2	12,1o,3,1	12,1o,1,9,7

Thus for both LRU 1 and LRU 2: $B^{(6)}_{(4,w)} \not\subseteq B^{(6)}_{(5,w)}$

3.4.3. Stack property of $A^{(t)} \subset MM^{(t)}$

$A^{(t)}$ is governed either by LRU 1 (in this case the stack property is obvious) or by LRU 2.

Theorem 6:

$A^{(t)}(i,w) \subset A^{(t)}(i+1,w)$ *for all* i, t *and* w.

Proof:

We restrict on LRU 2 - organization of $A^{(t)}$ and on $i > 2$.
For $i \leq 2$ the proof will be similar.
If t_0 is the first time when $A^{(t)}$ (consisting of i pages) will be full then obviously :

$$A^{(t)}(i,w) \subset A^{(t)}(i+1,w) \quad \text{for } t \leq t_0.$$

Now we proceed by induction on t to show the following:

$$A^{(t)}(i,w) = [q_1,\ldots,q_{i-2},a,b]$$

$$\longrightarrow \quad A^{(t)}(i+1,w) = [q_1,\ldots,q_{i-2},c,d,e] \quad \text{for } t \geq t_0$$

where

$$[c,d,e] = \begin{cases} [a,b,r] \\ [a,r,b] \\ [r,a,b] \end{cases} .$$

Three different cases will be considered:

I. $r_t \in A^{(t)}(i,w)$

In this case obviously: $A^{(t+1)}(i,w) \subset A^{(t+1)}(i+1,w)$

II. $r_t \in A^{(t)}(i+1,\omega)$, $r_t \notin A^{(t)}(i,\omega)$, i. e. $r_t = r$:

$$A^{(t+1)}(i+1,\omega) = [r,q_1,\ldots,q_{i-2},a,b]$$

$$A^{(t+1)}(i,\omega) = \begin{cases} [r,q_1,\ldots,q_{i-2},a] & \text{if } b \notin R_t \\ [r,q_1,\ldots,q_{i-2},b] & \text{if } b \in R_t \text{ , } a \notin R_t \\ [r,q_1,\ldots,q_{i-3},a,b] & \text{if } a,b \in R_t \end{cases}$$

III. $r_t \notin A^{(t)}(i+1,\omega)$

In dependence of the number e_A of the page which will be replaced from $A^{(t)}(i+1,\omega)$ we obtain:

1. $e_A = r$:

$$A^{(t+1)}(i+1,\omega) = [r_t,q_1,\ldots,q_{i-2},a,b]$$

$$A^{(t+1)}(i,\omega) = [r_t,q_1,\ldots,q_{i-3},c,d]$$

where $[c,d] \in ([q_{i-2},a],[q_{i-2},b],[a,b])$

2. $e_A = b$:

$$A^{(t+1)}(i+1,\omega) = \begin{cases} [r_t,q_1,\ldots,q_{i-2},a,r] \\ [r_t,q_1,\ldots,q_{i-2},r,a] \end{cases}$$

$$A^{(t+1)}(i,\omega) = [r_t,q_1,\ldots,q_{i-2},a]$$

3. $e_A = a$:

$$A^{(t+1)}(i+1,\omega) = \begin{cases} [r_t,q_1,\ldots,q_{i-2},b,r] \\ [r_t,q_1,\ldots,q_{i-2},r,b] \end{cases}$$

$$A^{(t+1)}(i,\omega) = [r_t,q_1,\ldots,q_{i-2},b]$$

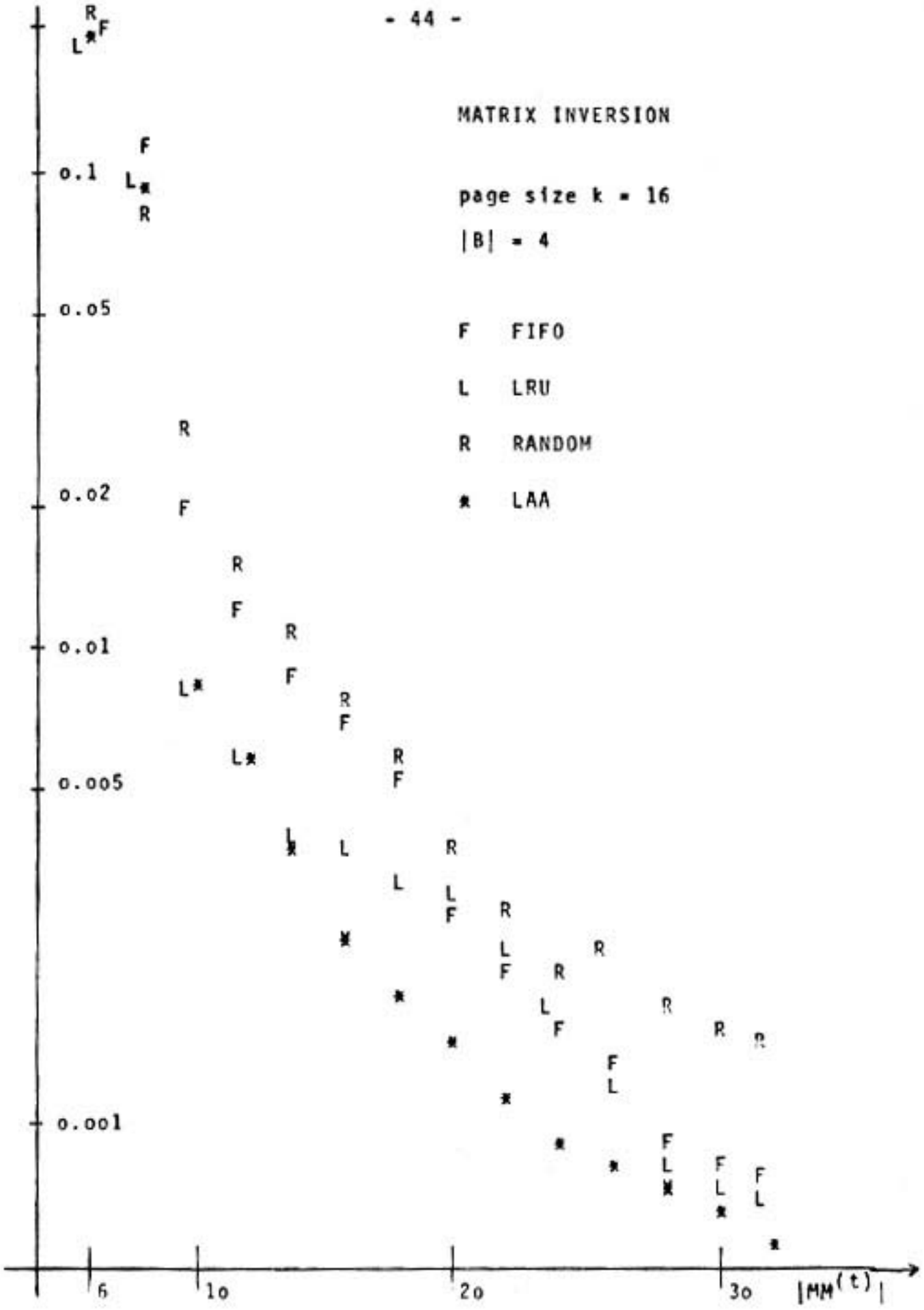
Appendix

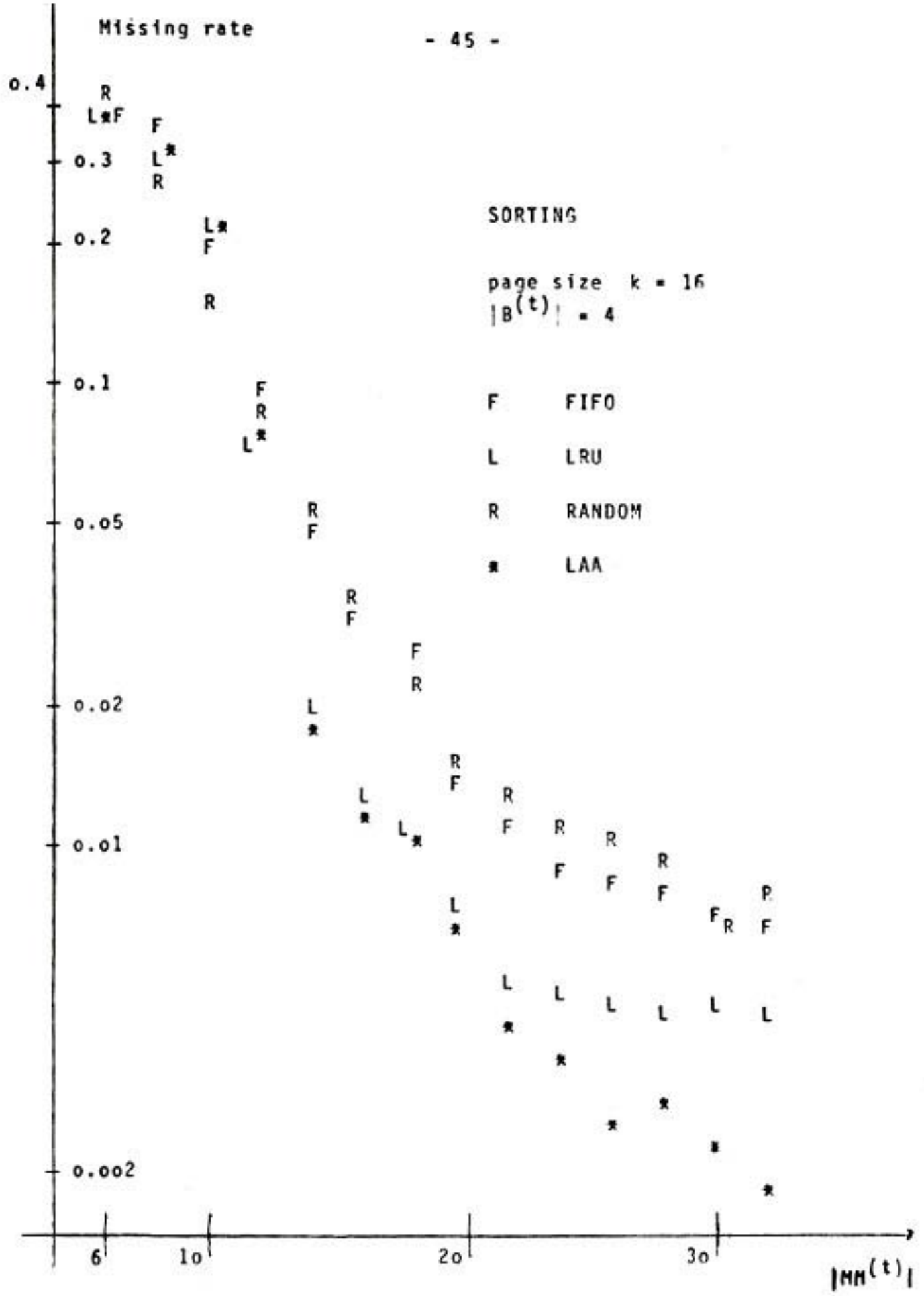
The performance of LAA has been compared with the behaviour of usual demand paging algorithms (RANDOM, FIFO, LRU). Simulation results for two programs (matrix inversion and sorting) are pictured in the following.
As a page size, k = 16 was choosen. Moreover the region I of instructions has been excluded from the simulations. If the page size is increased and/or the instruction code is included, the advantages of LAA would certainly be even more convincing. Finally, $B^{(t)}$ was restricted on 4 pages.

It will be remarked immediately that LAA behaves very bad if the size of MM is very small since in this case many active parts of the programs are dislocated by the pages which are loaded in advance.

On the other side, if there is a sufficient number of pages ($\geq$ 16 or $\geq$ 2o in our examples) the missing rate is reduced and LAA turns out to be favorable if h(i) < i for i = 2 and i = 3 (cf. definition 1).

Finally the dominance of LAA over demand paging algorithms will become clear from the fact that LAA speeds up the computation since look-ahead replacements may be carried out without interrupting the execution of the program.


MATRIX INVERSION
page size k = 16
|B| = 4
F FIFO
L LRU
R RANDOM
* LAA
0.1
0.05
0.02
0.01
0.005
0.001
6
10
20
30
|MM(t)|


Missing rate
0.4
0.3
0.2
0.1
0.05
0.02
0.01
0.002
SORTING
page size k = 16
|B(t)| = 4
F FIFO
L LRU
R RANDOM
* LAA
6
10
20
30
|MM(t)|

REFERENCES

[1] Aho, A. V., Denning, P. J. and Ullman, J. D.:
Principles of optimal page replacement.
J. ACM 18 (1971) 80 - 93.

[2] Baer, J.-L. and Sager, G. R.:
Measurement and improvement of program behaviour under paging systems.
In: Statistical computer performance evaluation.
Academic Press (1972) 241 - 264.

[3] Baer , J.-L. and Sager, G. R.:
On the dynamic definition of locality in virtual memory systems.
Colorado State University. Techn. Report CS 74-01.

[4] Belady, L. A.:
A study of replacement algorithms for virtual memory storage computers.
IBM Syst. J. 5 (1966) 78 - 101.

[5] Coffman, E. G. Jr. and Denning, P. J.:
Operating Systems Theory.
Prentice-Hall, Inc. (1973).

[6] Denning, P. J.:
The working set model for program behaviour.
Comm. ACM 11 (1968) 323 - 333.

[7] *Denning, P. J. and Schwartz, S. C.:*
Properties of the working set model.
Comm. ACM 15 (1972) 191 - 198.

[8] *Gelenbe, E., Lenfant, J. and Potier, D.:*
Analyse d'un algorithme de gestion simultanée Mémoire centrale - Disque de pagination.
Acta Informatica 3 (1974) 321 - 345.

[9] *Gibson, D. H.:*
Considerations in block-oriented systems design.
SJCC (1967) 75 - 80.

[10] *Hatfield, D. J. and Gerald, J.:*
Program restructuring for virtual memory.
IBM Syst. J. 10 (1971) 168 192.

[11] *Joseph, M.:*
An analysis of paging and program behaviour.
The Computer Journal 13 (1970) 48 - 54.

[12] *Lenfant, J.:*
The delay network model of program behaviour.
In: Computer architectures and networks.
North Holland (1974) 299 - 329.

[13] *Mattson, R.L., Gecsei, J., Slutz, D. R. and Traiger, I. L.:*
Evaluation techniques of storage hierarchies.
IBM Syst. J. 9 (1970) 78 - 117.

[14] *Muntz, R. R. and Opderbeck, H.:*
Stack replacement algorithms for two-level directly addressable paged memories.
SIAM J. Computing 3 (1974) 11 - 22.

[15] *Opderbeck, H. and Chu, W. W.:*
The page fault frequency replacement algorithm.
FJCC (1972) 597 - 609.