EXPLOITING BOTH SPATIAL AND TEMPORAL LOCALITY IN PAGE REPLACEMENT ALGORITHMS

By

XIANG HUI LIU

Bachelor of Science

Peking University

Beijing, P.R.China

1995

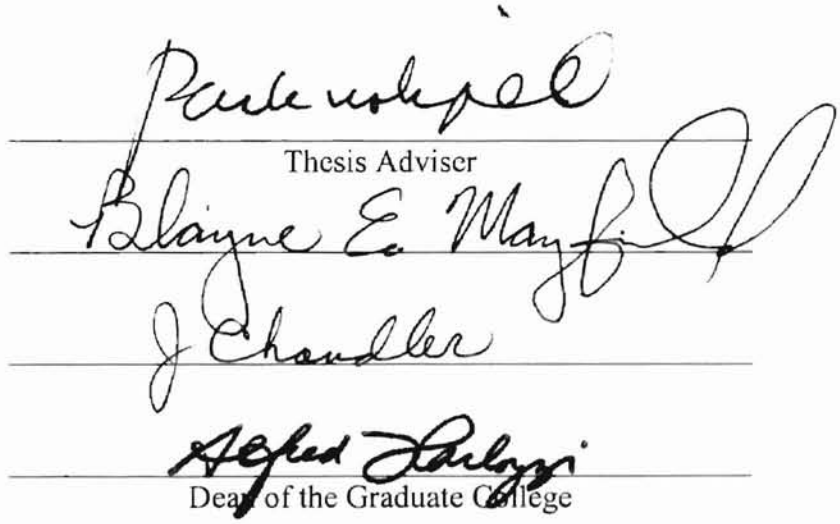
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
In partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2000

EXPLOITING BOTH SPATIAL AND TEMPORAL

LOCALITY IN PAGE REPLACEMENT

ALGORITHMS

Thesis Approved:



Thesis Adviser

Dean of the Graduate College

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor, Dr. Nohpill Park for his intelligent supervision, constructive guidance, inspiration and friendship. Without his selfless help I would not have my thesis done smoothly and timely. I am grateful to my other committee members Dr. John P. Chandler and Dr. Blayne E. Mayfield, whose guidance, assistance, encouragement, and friendship are also invaluable. Also I would like to express my sincere appreciation to Dr. H. K. Dai, who provided much good advice and assistance for this study.

Moreover, hearty thanks go to my family for their strong encouragement, endless love and support throughout this whole process. I also would like to give my special appreciation to my friends for their precious suggestions, invaluable assistance, understanding and encouragement.

Finally, I would like to thank all of you who helped me during this thesis study.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Since the technique of virtual memory was first introduced to allow larger programs than available physical memory [Silberschatz and Galvin 97], there have been a number of new mechanisms created to utilize the advantages of virtual memory to benefit the computer systems and the users, such as segmentation and paging. In a paging scheme, translation from logical addresses to physical addresses is necessary, and page faults occur when a virtual page referenced is not in memory. The selection of a page replacement algorithm, which is used to determine which page in memory should be removed to make room for currently referenced page, has a dramatic effect on the system performance since extra overhead will be needed due to page faults, thrashing may happen, and even slight improvements in paging schemes may yield large gains in system performance [Silberschatz and Galvin 97].

Although a wide variety of possible replacement algorithms exist, they can be grouped into two main categories. One is theoretical, which requires future knowledge of the entire sequence of memory references. Therefore it is impossible to implement in real time and can only serve as a tool of system analysis, such as Belady's MIN algorithm. The other is heuristic, which may rely on the history of reference strings.

The criterion to design a practical page replacement algorithm is to achieve the lowest page-fault rate [Silberschatz and Galvin 97]. LRU algorithm is one of the closest approximations to optimal replacement algorithm. The basic idea of LRU is to pick a page least recently used as the victim page on a memory miss. But the LRU algorithm has a flaw, in that it selects victim pages based on only the time of last reference. In general, it works well if the number of productive pages is small because the most recently used pages have a good chance of being in the favored subset of the memory [Martin et al. 90]. But once this fact changes, LRU becomes less effective. For example: given a reference string 0,1,2,3,0,1,2,3,0,1,2,3 and the memory size is 3, we can see that LRU has no advantage at all in terms of minimizing page-fault rate even though this reference string owns a very good locality property. Because of this observation, the purpose of this study is to mostly exploit both temporal and spatial locality without causing more memory traffic based on LRU algorithm. The idea is that on a memory miss, the selected victim page should be one which has relatively least locality relationship with the demanded page with the lowest reference time tag.

1.2 TERMINOLOGY

For the sake of clarity, a few terms which are used throughout the study are defined as follows.

- *Belady's Anomaly* reflects the fact that the page-fault rate may increase as the number of allocated frames increases for some page-replacement algorithms [Silberschatz and Galvin 97].

- *Independent reference model* assumes that the references are random and independent. In other words, reference strings in this model do not have any temporal and spatial locality [Jain 90].
- *Locality function* is a function with a single page A as a domain and a set of pages containing A with the size of *locality window size* as a range. It is used in the proposed extended LRU algorithm.
- *Locality window size* defines the number of pages involved in the range of *the locality function*.
- *Pure loops model* provides reference strings consisting of only loops. Within one loop, every reference is to distinct page and the distances of every two successive references are the same.
- *Victim page* corresponds to a page in memory that will be removed from memory on a memory miss.

1.3 LITERATURE REVIEWS

Since page references by computer programs usually exhibit good locality behavior [Jain 90], the stronger the locality, the more effective memory strategies [Martin et al. 90]. Several studies have concentrated on utilizing hardware techniques to better exploit temporal locality and/or spatial locality. Compared with set-associative caches, directed-mapped cache suffers from significantly higher miss rate [Jouppi 90]. In order to better exploit temporal locality and hence to reduce the miss ratio of directed-mapped caches, [McFarling 92] used a new hardware technique called dynamic exclusion and [Agarwal

and Pudar 93] used a technique of column-associative caches and by adding a rehash bit to each cache set, allow most of the conflicting data residing in the cache while the critical hit access path remains the same. For the same purpose, small fully-associative caches and prefetch buffers are used to support the mostly exploitation of spatial locality [Seznec 93].

Some page replacement algorithms require the future knowledge of the reference string. Belady's MIN algorithm is one of them. There are several research studies rely on Belady's MIN algorithm. [Burger et al. 96] focuses on pin bandwidth consideration for future microprocessors. It computes effective pin bandwidth and further estimate upper bound of effective pin bandwidth in terms of traffic inefficiencies defined and measured based on Belady's MIN policy. [Abraham and Sugumar 93] also explain accurately the notion of capacity and conflict misses by using Belady's MIN algorithm. Recently an extension of Belady's MIN algorithm is presented by [Temam 99]. It includes spatial locality into Belady's algorithm and exploits spatial and temporal localities simultaneously and optimally with miss ratio as a cost function. In this study, they not only formally prove that the extended algorithm can minimize miss ratios, but also experimentally show that the near-minimal memory traffic can also be achieved with a given amount of memory space. [Trivedi 76] defines an optimal demand prepaging algorithm by knowing the entire reference string in advance.

Since programs can't predict the future, an optimal replacement algorithm is unrealizable. LRU is a good approximation to optimal replacement algorithms [Hyde 88] and a number

of research studies are related to it. [Maruyama 75] proposes a MLRU page replacement algorithm by bringing M physically contiguous stored pages including the demanded page into the main memory on a miss instead of only one demanded page in LRU. [Martin et al. 90] compares the performances of LRU and its two modifications that are combinations of LRU and two different prefetching schemes. Under the assumption of independent reference model, [Van Den Berg and Gandolfi 92] has shown that LRU is better than FIFO. Furthermore, [Chrobak and Noga 98] proves a conjecture that the competitive ratio of LRU on each access graph is not more than the competitive ratio of FIFO. [O'Neil et al. 93] designs a new page replacement policy called LRU-K method by analyzing the times of the last k references to memory pages. And the optimality of this algorithm is proved further using the independent reference model [O'Neil et al. 99].

CHAPTER 2

PRELIMINARIES

2.1 VIRTUAL MEMORY

When people first encountered programs which had larger size than available memory size many years ago, the solution usually adopted was to split the program into pieces that were called overlays [Tanenbaum 92]. The overlays were kept on the disk and swapped in and out of memory by the operating system. Since the generation of overlays done by the programmer was very time consuming, boring, and complex, a way to overcome this drawback was soon developed, which is known as virtual memory [Silberschatz and Galvin 97].

The basic idea behind virtual memory is that it allows the execution of processes that may not be completely in memory [Silberschatz and Galvin 97]. So program size is no long a constraint of programmers and multiprogramming/multitasking become feasible. The operating system keeps those parts of the program currently in use in main memory, and the rest on the secondary storage [Tanenbaum 92].

In any virtual memory system, there are two kinds of addresses for programs: virtual address and actual physical memory address. Virtual addresses are produced by programs and form the virtual address space. Physical memory addresses are the actual physical addresses in memory.

2.2 PAGING

There are two common techniques used in virtual memory system: paging and segmentation. In a segmentation scheme, the virtual address space is divided into many segments with variable sizes and independent address spaces. External fragmentation exists [Hyde 88]. In a paging scheme, the virtual address space is partitioned into units with fixed size called pages [Tanenbaum 92]. The units in physical memory that correspond to pages of virtual memory space have the same size and are called page frames [Carr 84]. Internal fragmentation exists [Hyde 88].

A hardware unit called memory management unit (MMU) is used to translate the virtual addresses onto the physical memory addresses. Figure [1] shows the architecture of the address translation [Moote 89].

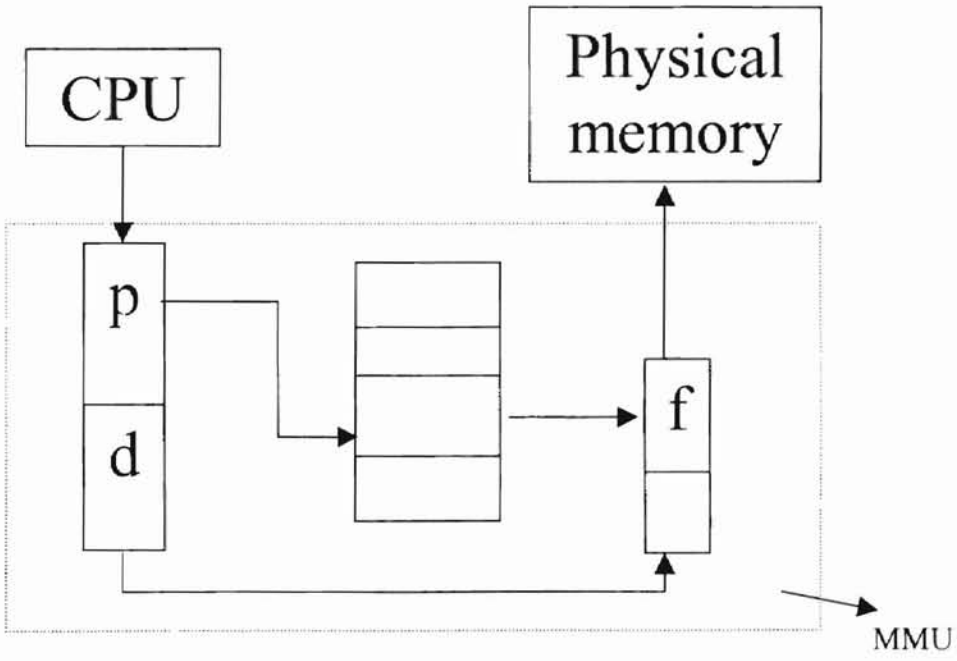


Figure 1. Address Translation Architecture

When a virtual address is sent to the MMU, MMU determines the page number(p) to which the virtual address belongs, gets the corresponding page frame number(f) according to the page table, translate the virtual address (| p | d |) into the physical address (| f | d |), and outputs it onto the bus.

If the MMU notices that the page is unmapped, then it will cause the central processing unit (CPU) to trap to the operating system. This trap is called a page fault [Silberschatz amd Galvin 97] . Upon the page fault occurs, the operating system will pick a page frame(called victim page) using an algorithm, writes its content back to the disk if it is modified, fetches the page just referenced into the page frame just freed, changes the map in the page table, and restarts the trapped instruction. Obviously, the occurrence of page faults must cause the slowdown of the system execution and take extra overhead to be resolved. The worse thing is that if a heavily used page is evicted and it will probably have to be brought back in quickly, the thrashing may happen.

2.3 LOCALITY OF MEMORY REFERENCES

Locality is the property that references to program tend to cluster into groups in time and/or space [Thoreson and Long 89] [Denning 72] [Denning 70]. A high degree of locality ensures that less number of page faults. Locality can be differentiated into two classes: temporal locality and spatial locality.

[Thoreson and Long 89] states that temporal locality is with respect to time clustering for a set of pages. That is, if a set of pages are referenced during a time interval, it is likely that they will be referenced again during the immediately following time interval [Schneider 83]. Temporal locality is well exhibited in short loops.

Spatial locality occurs when two successive references have adjacent virtual spaces. In other words, if word w is referenced at time t, then words in the range w-i to w+i for some small i are likely to be referenced at time t+1 [Spirn 77]. In a sequential environment, straight-line code usually produces good spatial locality [Thoreson and Long 89].

2.4 PAGE REPLACEMENT ALGORITHMS

Page replacement algorithms are used to determine which pages are removed from the memory when page faults occur.

Optimal Replacement Algorithm:

Needless to say, on a page fault, the best choice of the victim page is the virtual page that the program will not reference for the longest period time [Tanenbaum 92]. This is called the optimal replacement algorithm. [Belady 66] proposed one called MIN algorithm based on the future knowledge of the entire sequence of page references for a program. The objective is to determine a minimum sequence of pages needed to be removed and provide the minimum number of memory misses. The implementation of the algorithm is

a two-pass job. The first pass is to store the program's complete sequence of references. And the second one is to reconstruct a minimum-replacement sequence by working backward on the program trace. This algorithm guarantees the minimum miss ratio when temporal locality is exploited optimally and it can be used for general system studies and specifically for an evaluation of the heuristic replacement algorithms [Temam 99].

LRU Algorithm:

This algorithm is based on the observation that the recent past is a good indicator of the immediate future [Tanenbaum 92]. When a page fault occurs, a page selected to be evicted from the memory is one that is least recently used. The common program structures supporting this observation are loops, subroutine calls , etc. LRU usually can provide very low memory miss ratio, however, its implementation requires considerable hardware and software overhead since there is a time stamp associated to each reference.

LFU Algorithm:

LFU is an approximation to LRU. Instead of having a time tag to each reference, each page is attached a frequency count of references which is incremented periodically [Hyde 88]. The page removed from the memory is one with the lowest frequency count. The overhead of LFU is lower than LRU, but the problem is that heavy frequency of a page in initialization phrase may mislead the victim selection in program execution and a page with potential heavy references may be evicted immediately after it is brought into the memory since its frequency count is just 1 at that time.

FIFO Algorithm:

A data structure, queue, is used to keep all the pages in the system in the implementation of the algorithm. On a page fault, it simply replaces the page at the head of the queue and adds it to the tail of the queue. The FIFO algorithm is easy to implement and has low overhead, but it is lacking of locality consideration and causes Belady's anomaly. So it is rarely used.

CHAPTER 3

PROPOSED EXTENDED LRU ALGORITHM

3.1 OVERVIEW OF ALGORITHM

The proposed algorithm is a run-time extended LRU algorithm that exploits both spatial and temporal locality. A *locality window* is used in the proposed algorithm, which contains pages likely to be referenced next due to the spatial and/or temporal locality considerations. The basic idea is that on a memory miss, the selected victim page should be the one which has relatively least locality relationship with the demanded page with the lowest reference time tag, i.e., outside the locality window with the lowest reference time tag.

There are three parameters to characterize the proposed algorithm:

(1) memory size denoted by M_s defines the number of allocated frames.

(2) locality window size denoted by W_s gives how many pages are involved in the locality window.

(3) locality function $F: A \rightarrow \{A_1, A_2, ..., A_{W_s}\}$, where $\exists i \ni A_i = A$ and A is the current page referenced, defines what pages are involved in the locality window.

For example, if a locality function F is defined as

$A \rightarrow \{A - \left\lfloor \frac{w_s - 1}{2} \right\rfloor, ..., A-1, A, A+1, ..., A + \left\lceil \frac{w_s - 1}{2} \right\rceil\}$, $W_s = 4$, and $A = 5$, then the

locality window contains pages 4, 5, 6, and 7.

As in the LRU algorithm, we need to keep track of the time of each reference to implement the new algorithm.

3.2 DETAILS OF PROPOSED ALGORITHM

Definition: a loop mentioned in the proposed algorithm means $x_1, x_2, ..., x_k, x_1, ...$ where

$d = x_j - x_{j-1}$ is constant for $2 \le j \le k$.

How to tell if the reference string is forming a loop: let the current page be x_i where $1 \le i \le k-1$, if the reference string up to the current page is $..., x_1, x_2, ..., x_k, x_i$ and $x_j - x_{j-1}$ is constant for $i+1 \le j \le k$, then there exists a loop and its iteration is $x_i, ..., x_k$.

How to tell if a loop ends: let the current page be y_i and current loop iteration be $x_i, ..., x_k$, if y_{i-1} (previous reference) is equal to x_t for $i \le t \le k$ but y_i is not equal to x_{t+1} if $t+1 \le k$ or x_i if $t+1 > k$, then the loop ends.

Two parameters in the proposed algorithm, locality window size W_s and locality function F, together determine the locality window thereby significantly affect the selection of a victim page. Since different reference strings exhibit various locality behaviors, in order

to exploit both spatial and temporal locality mostly and efficiently, they are adaptive during the algorithm's implementation.

By the definition of spatial locality: if a word w is referenced at time t, then a word in the range $w-i$ to $w+i$ for some small i is likely to be referenced at time $t+1$ [Spirn 77]. So if the locality window size is W_s, then in order to exploit the spatial locality, initially, in the proposed algorithm, F is defined as $A \rightarrow \{A-\left\lfloor \frac{w_s-1}{2} \right\rfloor,..., A-1, A, A+1,..., A+\left\lceil \frac{w_s-1}{2} \right\rceil\}$. Hence locality window contains pages which are spatially close to the current page and locality relationship here means the spatial locality relationship with the current page and every page outside the window is considered spatially far away from it and serves as a candidate of a victim page. Temporal locality means that if a set of pages are referenced during a time interval, it is likely that they will be referenced again during the immediately following time interval [Schneider 83]. Once the reference string up to and including the current page is forming a loop, then W_s is set to be the length of the loop minus one and F is adjusted to be $A \rightarrow \{A, A+d,..., A+(W_s-1)d\}$ under the assumption that this loop would last for a while. And now, pages in the locality window are temporally close to the current page and temporal locality is the major concern. Once the loop ends, W_s and F will be reset back to the initial values.

Since the locality window contains pages which are likely referenced next, the victim page should be outside the window and have the lowest time tag in order to both exploit spatial and temporal locality.

Principle of the proposed algorithm:

Initially, the locality window size is W_s and locality function F :

$A \rightarrow \{A - \left\lfloor \frac{w_s - 1}{2} \right\rfloor, ..., A, ..., A + \left\lceil \frac{w_s - 1}{2} \right\rceil\}$ which gives W_s pages including page A around page A.

Step (1): **IF** it is not the end of the reference string, **THEN**

IF there is a loop existing, **THEN** go to step (3),

ELSE, go to step (2).

ELSE, stop.

Step (2): On a miss, **IF** the reference string is forming a loop, **THEN**

set $W_s = L - 1$ where L is the length of one loop iteration, F :

$A \rightarrow \{A, A + d, ..., A + (W_s - 1)d\}$ where d is the distance of two successive references in the loop. And pick_victim().

Go to Step (4).

Step (3): **IF** a loop ends, **THEN** reset W_s and F back to initial values.

On a miss, pick_victim().

Go to Step (4).

Step (4): Process the next reference, go to step (1).

Pick_victim(): pick a page which is not in the locality window and has the lowest time tag as the victim page if any. If every page is in the window, then pick one with the lowest time tag. Remove the selected page, set the time tag of the referenced page the current time, and add the new page into memory.

Step (2) is to adjust the locality window size to mainly exploit temporal locality behavior exhibited by the reference string up to and including the demanded page upon the detection of the existence of a loop. Claim 6 in Section 5.4 will explain why it is adjusted to this value. For example, if a reference string is 1,3,4,0,2,4,6,0,2,4,6,5, then at the reference to the third page 1, W_s is set to be 3 and $F : A \rightarrow \{A, A+2, A+4\}$. Step (3) is to restore the initial values of W_s and F once the loop ends in an attempt to exploit spatial locality. For the same reference string as above, at the reference to page 5, W_s and F are reset to initial values since the loop ends there.

In the LRU, at the moment of a page fault, the selection of a page to be evicted from memory depends only on its time tag, whereas in the proposed extended LRU, it not only depends on the time tag but also is related to the pages in the current locality window.

3.3 EXAMPLES

Let memory size $M_s = 4$ and a reference string be 0, 1, 2, 3, 4, 0, 1, 0, 5, 1, 2, 3, 4, 5.

M1-M4 in the following tables stand for 4 memory slots.

Applying LRU algorithm:

String:	0	1	2	3	4	0	1	0	5	1	2	3	4	5
M1	0	0	0	0	4	4	4	4	4	4	2	2	2	2
M2		1	1	1	1	0	0	0	0	0	0	3	3	3
M3			2	2	2	2	1	1	1	1	1	1	1	5
M4				3	3	3	3	3	5	5	5	5	4	4
Page faults	*	*	*	*	*	*	*		*		*	*	*	*

Table I: Results Of LRU Algorithm

The total number of page faults = 12.

Applying proposed extended LRU algorithm:

Initially, let $W_s = 3$ and then F be $A \rightarrow \{A-1, A, A+1\}$.

$W_s = 3$, window={2,3,4} $W_s = 4$, window={0,1,2,3} $W_s = 3$, window={-1,0,1}

	0	1	2	3	4	0	1	0	5	1	2	3	4	5
M1	0	0	0	0	4	0	0	0	0	0	2	2	2	2
M2		1	1	1	1	1	1	1	1	1	1	1	4	4
M3			2	2	2	2	2	2	5	5	5	5	5	5
M4				3	3	3	3	3	3	3	3	3	3	3
Page faults	*	*	*	*	*	*			*		*		*	

Table II: Results Of Proposed Extended LRU Algorithm

The total number of page faults = 9.

Applying OPT algorithm:

	0	1	2	3	4	0	1	0	5	1	2	3	4	5
M1	0	0	0	0	0	0	0	0	5	5	5	5	5	5
M2		1	1	1	1	1	1	1	1	1	1	3	3	3
M3			2	2	2	2	2	2	2	2	2	2	2	2
M4				3	4	4	4	4	4	4	4	4	4	4
Page faults	*	*	*	*	*				*			*		

Table III: Results Of OPT Algorithm

The total number of page faults = 7.

Comparing the results of three algorithms, the proposed algorithm performs much better than LRU. In the meanwhile, the reference string used in above example exhibits good spatial locality behavior.

3.4 STACK ALGORITHM

Claim 1: The proposed extended LRU algorithm is a stack algorithm.

By [Mattson et al. 70], A replacement algorithm is called a stack algorithm if the memory contents satisfies the inclusion property for every page trace and every point in time. Given a page trace $x_1, x_2, ..., x_L$ and any time t (where $1 \le t \le L$), let C stand for the memory capacity, $B_t(C)$ stand for the set of pages in memory just after the completed

reference to x_t, initially, $B_0(C) = \phi$, D_t stand for the set of distinct pages references in $x_1, x_2, ..., x_L$, and γ_t stand for the number of pages in D_t, the inclusion property is: $B_t(1) \subseteq B_t(2) \subseteq ... \subseteq B_t(\gamma_t) = B_t(\gamma_t + 1) = ...$ where $|B_t(C)| = C$ for $1 \le C \le \gamma_t$ and $|B_t(C)| = \gamma_t$ for $C \ge \gamma_t$.

Proof: In order to prove this claim, it suffices to prove that the algorithm satisfies $B_t(C) \subseteq B_t(C+1)$ for any $0 \le t \le L$ and $1 \le C < \gamma_t$.

Prove by mathematical induction on the time t.

(1) $t = 0$, by the definition, $B_0(C) = \phi$ and $B_0(C+1) = \phi$, so $B_0(C) \subseteq B_0(C+1)$.

$t = 1$, obviously, $B_1(C) = x_1$ and $B_1(C+1) = x_1$, so $B_1(C) \subseteq B_1(C+1)$.

(2) Assume that $B_t(C) \subseteq B_t(C+1)$ for $\forall t \ge 0$.

Then for $t+1$, the referenced page is x_{t+1}.

(i) If $|B_t(C)| < C$ and $|B_t(C+1)| < C+1$, then $B_{t+1}(C) = B_t(C) \cup \{x_{t+1}\}$, and $B_{t+1}(C+1) = B_t(C+1) \cup \{x_{t+1}\}$, so $B_{t+1}(C) \subseteq B_{t+1}(C+1)$.

(ii) If $|B_t(C)| = C$ and $|B_t(C+1)| < C+1$, then $B_{t+1}(C) = (B_t(C) - \{x_i\}) \cup \{x_{t+1}\}$ for some x_i, and $B_{t+1}(C+1) = B_t(C+1) \cup \{x_{t+1}\}$, so $B_{t+1}(C) \subseteq B_{t+1}(C+1)$.

(iii) If $|B_t(C)| = C$ and $|B_t(C+1)| = C+1$, then there are three cases for x_{t+1}:

Case 1: $x_{t+1} \in B_t(C)$, hence, $x_{t+1} \in B_t(C+1)$ since $B_t(C) \subseteq B_t(C+1)$.

Case 2: $x_{t+1} \notin B_t(C)$, but $x_{t+1} \in B_t(C+1)$.

Case 3: $x_{t+1} \notin B_t(C+1)$, hence, $x_{t+1} \notin B_t(C)$ since $B_t(C) \subseteq B_t(C+1)$.

Case 1: since x_{t+1} belongs to both sets, there is no page faults occurred. By the algorithm, $B_t(C) = B_{t+1}(C)$ and $B_t(C+1) = B_{t+1}(C+1)$, so $B_{t+1}(C) \subseteq B_{t+1}(C+1)$

Case 2: For the same reason, $B_t(C+1) = B_{t+1}(C+1)$.

Need to pick a victim page from $B_t(C)$ for x_{t+1}. No matter which page is selected, it has to reside in $B_t(C)$, say x_i where $1 \le i \le t$, so $B_{t+1}(C) = (B_t(C) - \{x_i\}) \cup \{x_{t+1}\}$.

$B_t(C) \subseteq B_t(C+1) = B_{t+1}(C+1)$, $(B_t(C) - \{x_i\}) \subset B_t(C)$, and $x_{t+1} \in B_t(C+1)$

so $B_{t+1}(C) = (B_t(C) - \{x_i\}) \cup \{x_{t+1}\} \subseteq B_{t+1}(C+1)$, i.e. $B_{t+1}(C) \subseteq B_{t+1}(C+1)$.

Case 3: Let $B_t(C) = \{y_1, ..., y_C\}$ and $B_t(C+1) = \{y_1, ..., y_C, y_{C+1}\}$ where $y_m \in \{x_1, x_2, ..., x_t\}$ for $\forall 1 \le m \le C+1$, let y_i be the victim page from $B_t(C)$, let y_j be the victim page from $B_t(C+1)$.

If $j = C+1$, then no matter what y_i is, $B_{t+1}(C) \subseteq B_{t+1}(C+1)$.

If $j \le C$, then

(a) if y_j is in the window $F(x_{t+1})$, then $\forall m \ne j, 1 \le m \le C+1$, the time tag of $y_j <$ the time tag of y_m, certainly it is true for $\forall m \ne j, 1 \le m \le C$, so $y_i = y_j$, therefore, $B_{t+1}(C) \subseteq B_{t+1}(C+1)$ since $B_t(C) \subseteq B_t(C+1)$.

(b) if y_j is not in the window $F(x_{t+1})$, then $\forall m \ne j, 1 \le m \le C+1$, y_m either is in the window or the time tag of $y_j <$ the time tag of y_m. Since the window for C case

and $C+1$ case are the same, certainly it is true for $\forall m \neq j, 1 \leq m \leq C$, so $y_i = y_j$, therefore, $B_{t+1}(C) \subseteq B_{t+1}(C+1)$ since $B_t(C) \subseteq B_t(C+1)$.

Therefore, $B_{t+1}(C) \subseteq B_{t+1}(C+1)$ for $t+1$.

So claim is true. That is, the proposed extended LRU algorithm is a stack algorithm.

Since the proposed extended LRU algorithm is a stack algorithm, it will not suffer from Belady's Anomaly and hence it will be able to be predictable and further improved.

3.5 PERFORMANCE ANALYSIS

Definition: a pattern is a substring of the reference string in which the distances of every two references are the same. For example, 1,3,5,7,9,11,13,15.

Analysis on time complexity:

Let N be the length of the reference string up to the current reference, M be the memory size, i.e., the number of frames, and L be the maximum length of pattern in the reference string.

Without loss of generality, we assume that $N > M$.

- The proposed algorithm:

To detect if a reference is a hit, it needs to check every page in memory, so the time needed is $O(M)$,

To determine if a reference string is forming a loop, it needs to go through the pattern, so time needed is $O(L)$,

To select a victim page from memory, the needed time is $O(M)$,

The other operations such as assignment, addition, etc need time $O(1)$.

Since once an environment is given, M is constant, so $O(M) = O(1)$.

Therefore, total time complexity $= O(NL)$.

- The LRU algorithm:

To detect if a reference is a hit, it needs to check every page in memory, so the time needed is $O(M)$,

To select a victim page from memory, the needed time is $O(M)$,

The other operations such as assignment, addition, etc need time $O(1)$.

Therefore, total time complexity $= O(N)$.

- Comparison:

By the definition of a pattern in the proposed algorithm, obviously, $L \leq$ the number of distinct pages in the reference string.

The worst case for the proposed algorithm compared with LRU is that every reference is to a distinct page and the distance of every two successive references are the same, then $L = N$ and the time complexity is $O(N^2)$. So the proposed algorithm has some extra time overhead in comparison with LRU.

Since pages in a pattern must be distinct and evenly distributed, generally, $L << N$ and $O(LN) \sim O(N)$.

So the proposed algorithm and LRU have the same time complexity of $O(N)$ on the average.

Analysis on space complexity:

Let N be the length of the reference string up to the current reference, M be the memory size, i.e., the number of frames, and L be the maximum length of pattern in the reference string.

Without loss of generality, we assume that $N > M$.

- The proposed algorithm:

Each page has a time of last reference associated and it is one of factors affecting the selection of a victim page.

To keep M pages information in memory, the space needed is $O(M)$,

The pattern used in the algorithm needs space $O(L)$.

Space of $O(1)$ is needed for the other operations.

Therefore, total space complexity $= O(L)$.

- The LRU algorithm:

The selection of a victim page in LRU is based on only the time of last reference, so the space of $O(M)$ is necessary to store pages information.

The other operations need space $O(1)$.

Therefore, total space complexity $= O(1)$.

- Comparison:

By the similar discussion to comparison of time complexity, the worst case of space complexity of the proposed algorithm is $O(N)$ but it is approximately $O(1)$ on the average.

Analysis on the number of page faults:

Definition: The page-fault rate is the number of page faults in a reference string divided by its length, which is denoted by PFRate.

Notations:

- PLM: *pure loops model*. For example: $x_1, x_2, ..., x_k, x_1, x_2, ...x_k, ...$
- PLM_kn: a pure loops model with n loop iterations and k distinct pages in each loop iteration, i.e. $x_1, x_2, ..., x_k, x_1, x_2, ...x_k, ..., x_1, x_2, ..., x_k$ with the length of nk.
- NumPF: the total number of page faults for a reference string.
- N_{PF}: the number of page faults in one iteration, i.e., $x_1, x_2, ..., x_k$.
- H_f: the number of hits in $x_1, x_2, ..., x_w$.
- H_m: the number of hits in $x_{w+1}, ..., x_{(n+1)w}$ where $n = \left\lfloor \frac{k}{w} \right\rfloor - 1$.
- H_l: the number of hits in $x_{nw+1}, ..., x_k$ where $n = \left\lfloor \frac{k}{w} \right\rfloor$.

Claim 2: For any reference string in PLM_kn, if memory size is m and the locality window size set in the proposed algorithm is w, then the total number of page faults will be

$$NumPF = \begin{cases} k, & \text{if } k \le m, \\ k + N_{PF}(n-1), & \text{otherwise,} \end{cases}$$

where

$$N_{PF} = \begin{cases} k - (H_f + H_m + H_l), & \text{if } m + w - k > 0, \\ k, & \text{otherwise.} \end{cases}$$

$$H_f = \min(m + w - k, m - 1),$$

$$H_m = \sum_{i=1}^{\left\lfloor \frac{k}{w} \right\rfloor - 1} H,$$

$$H = \begin{cases} m + w - k, & \text{if } i \le \left\lfloor \dfrac{m-1}{w} \right\rfloor, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$H_l = \begin{cases} m - \left\lfloor \dfrac{k}{w} \right\rfloor w, & \text{if } \left\lfloor \dfrac{k}{w} \right\rfloor = \left\lfloor \dfrac{m-1}{w} \right\rfloor \text{ and } w < k, \\ 0, & \text{otherwise.} \end{cases}$$

Proof: See Appendix A.

Claim 3: For any reference string in PLM_kn, let memory size be m, then the page-fault rate in LRU is

$$PFRate = \begin{cases} 1/N, \text{ if } k \le m, \\ 100\%, \text{ otherwise.} \end{cases}$$

Proof: See Appendix B.

Claim 4: For any reference string in PLM_kn, if the locality window size w is greater than or equal to $k-1$, then the number of page faults in proposed algorithm is independent of w.

Proof: See Appendix C.

Claim 5: N_{PF} given in Claim 2 is a non-increasing function of the locality window size w.

Proof: See Appendix D.

Claim 6: For any reference string in PLM_kn, the locality window size w of $k-1$ is optimal in terms of minimizing the number of page faults in proposed algorithm.

Proof: Directly from Claim 4 and Claim 5.

Claim 7: Initial locality window size has no effect on the number of page faults for any reference string in PLM_kn in the proposed algorithm.

Proof: See Appendix E.

From Claim 6, the locality window size in the proposed algorithm is set to be the length of loop -1 in an attempt to optimally exploit spatial locality of reference string. From Claim 3, it is shown that for any reference string in the pure loops model denoted by PLM_kn, as long as k is greater than the memory size, the number of page faults in LRU will be the number of references. So this model is the worst case for LRU. However, reference strings in this model exhibit good spatial and temporal locality. From Claim 2, it is shown that the proposed algorithm performs much better than LRU under this model. In the proposed algorithm, the locality window size is set to $k-1$ once there exists a loop

where k is the number of distinct pages in this loop. For a reference string in the PLM_kn model, the locality window size of $k-1$ will be set at the second reference to the first page within the loop. Without loss of generality, we can assume that $k > m$ since when $k \le m$ every algorithm achieves k page faults. By substituting w by $k-1$ in the formula given in Claim 2, we will get $N_{PF} = k - m + 1$ and the ratio of number of page faults in the proposed algorithm and LRU is

$\frac{k + (n-1)(k-m+1)}{kn} = \frac{1}{n} + \frac{k-m+1}{k} - \frac{k-m+1}{kn}$. As $n \to \infty$, the ratio will approach $\frac{k-m+1}{k} < 1$. So for any reference string in the PLM or its substring, the proposed algorithm must achieve less number of page faults than LRU.

On the other hand, it is possible that pages removed from memory due to locality consideration in the proposed algorithm will be referenced next and they remain in memory in LRU algorithm. In this case, we can assume that there exists no loop since the proposed algorithm must outperform LRU within a loop. Then the reference string must exhibit bad spatial locality after the memory is full. The reason is because in the proposed algorithm, the locality window contains the current page A, pages $\{A-1,..., A-\left\lfloor \frac{w_s - 1}{2} \right\rfloor\}$, and pages $\{A+1,..., A+\left\lceil \frac{w_s - 1}{2} \right\rceil\}$ and according to the principle of the proposed algorithm the victim page must be outside the window, hence it can not be spatially close to the current page. For example, given a reference string 1,2,3,4,1,4,2,4 and memory size 3, it turns out that the page-fault rate in LRU is 75% and 100% in proposed algorithm by applying them to this reference string. The numbers of

page faults are the same up to the first reference to page 4 between these two algorithms. The difference arises from the second reference to page 1. In the meanwhile, the reference string exhibits bad spatial locality from then on.

Consequently, LRU achieves bad performance for a reference string with a large number of productive pages and the proposed algorithm alleviates this problem. If a reference string is in the pure loops model, the proposed algorithm must produce no more number of page faults than LRU in the worst case and on the average much less than LRU. For random reference strings, they are comparable. However, if a reference string exhibits good spatial and temporal locality, then generally proposed algorithm should perform not worse than LRU.

CHAPTER 4

SIMULATIONS

4.1 METHODOLOGY

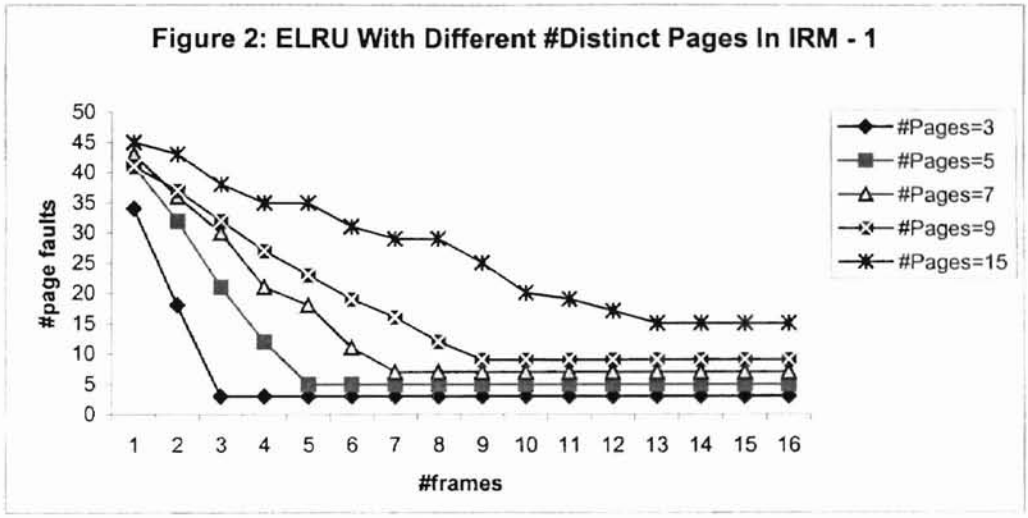
The simulations are based on two models: *independent reference model* and *pure loops model*. Independent reference model assumes that the references are random and independent. In other words, reference strings in this model do not have any temporal or spatial locality [Jain 90]. On the other hand, reference strings in the pure loops model are sequences of loops. They exhibit very good locality behavior.

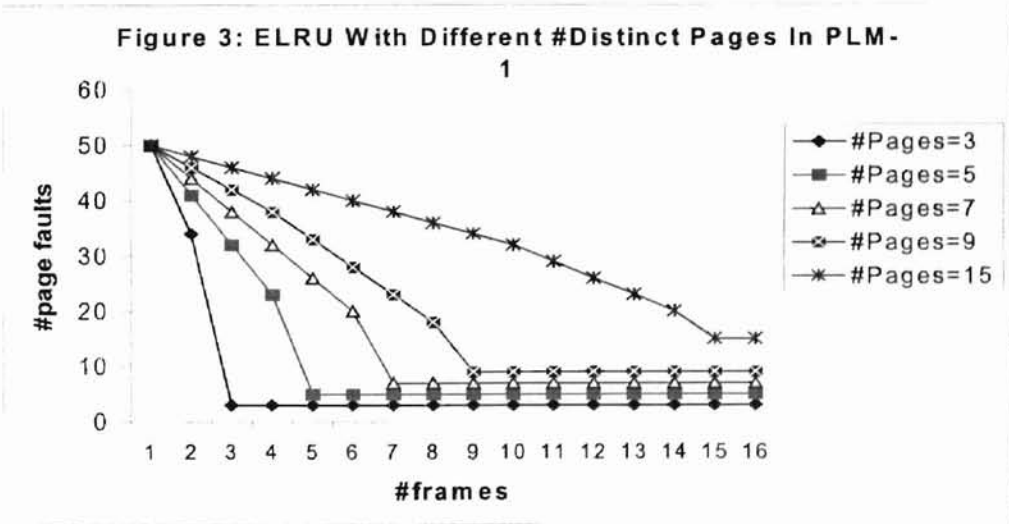
Five algorithms: OPT, LRU, LFU, FIFO, and proposed extended LRU algorithm are simulated and compared in terms of a traditional metric for performance of memory replacement algorithm, i.e., the number of page faults.

The proposed algorithm is characterized by three parameters: memory size, locality window size, and locality function. Since the algorithm is a stack algorithm, the larger the memory size, the less the number of page faults. To confirm this result, simulations with different memory sizes are conducted. Locality window size gives the range of the locality consideration and locality function defines the specific addresses involved in the locality consideration. Both locality window size and locality function affect the selection of victim pages and the number of page faults.

4.2 RESULTS OF SIMULATIONS

Notation: ELRU stands for the proposed extended LRU algorithm.





Figures [2] and [3] are based on simulations of five reference strings in the independent reference model and pure loops model. In this simulation, the initial locality window size

is set to 3 and the length of reference strings is set to 50 randomly. The figures show that the number of page faults is a non-increasing function of the number of frames. Moreover, it is a strictly decreasing function before it hits the number of pages in the pure loops model. Thus, The correctness of Claim 1 is verified. That is, the proposed algorithm is a stack algorithm. In addition, figures indicate that the number of page faults never exceed the number of pages if the number of allocated frames is no less that it. And the less the number of pages, the faster the number of page faults decreases as the number of frames increases in the pure loops model.

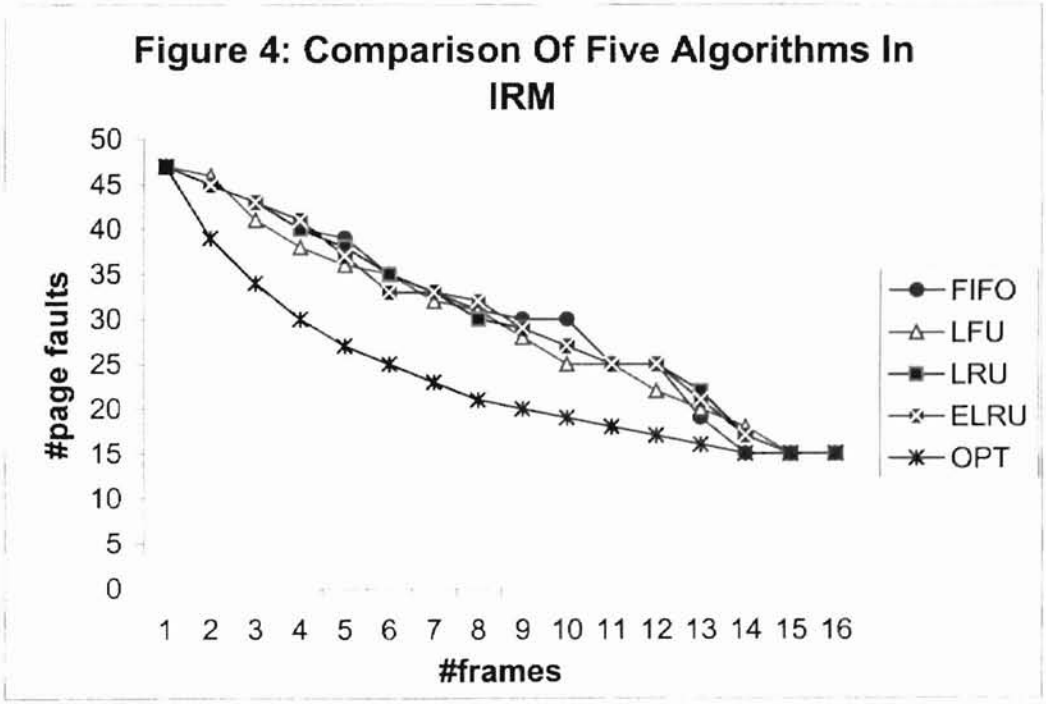


Figure [4] shows the comparison of the performance of the five algorithms using one reference string in the independent reference model. In this simulation, the number of

pages is set to 15, the initial locality window size is set to 3, and the length of reference string is set to 50 randomly. Apparently, OPT has the lowest number of page faults in all cases of number of frames. The performances of other four algorithms fluctuate.

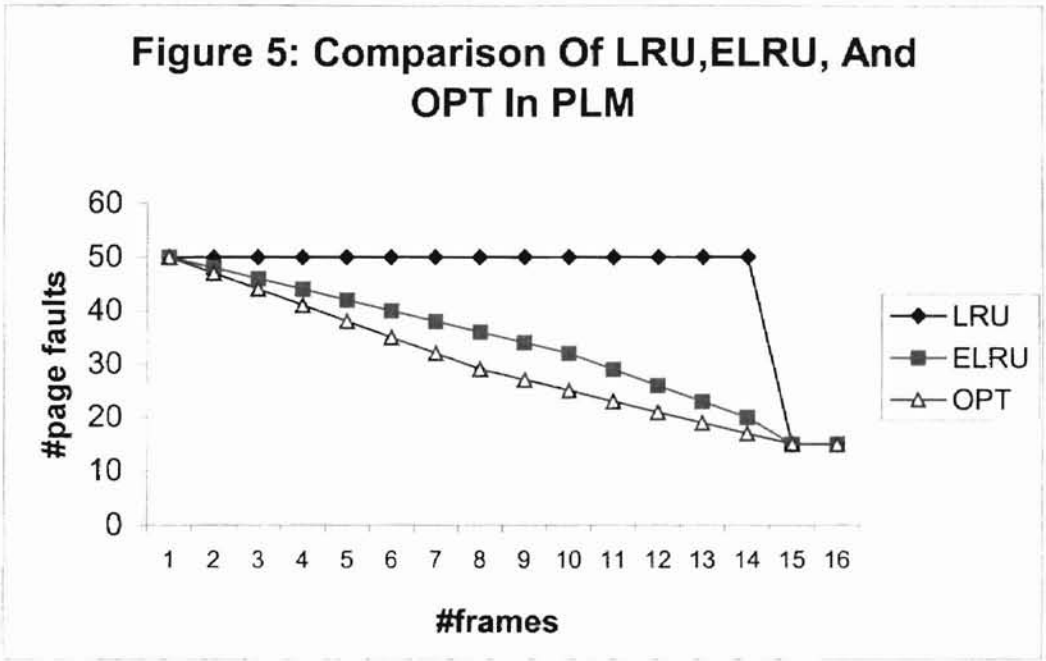


Figure [5] is to compare the performances of three algorithms using one reference string in the pure loops model. In this simulation, the number of pages is set to 15, the initial locality window size is set to 3, and the length of reference string is set to 50 randomly. It is shown that under this model, ELRU performs much better than LRU and is even closer to OPT. Before the number of allocated frames achieves the number of pages in the reference string, the more the number of frames, the better ELRU than LRU, whereas all of them cause the same number of page faults after that.

Figure 6: ELRU With Different #Distinct Pages In IRM - 2

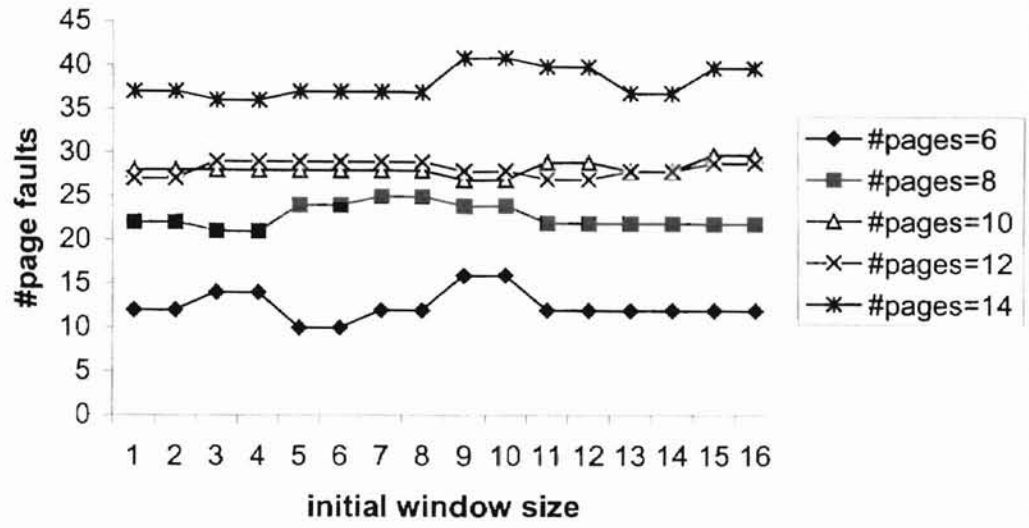
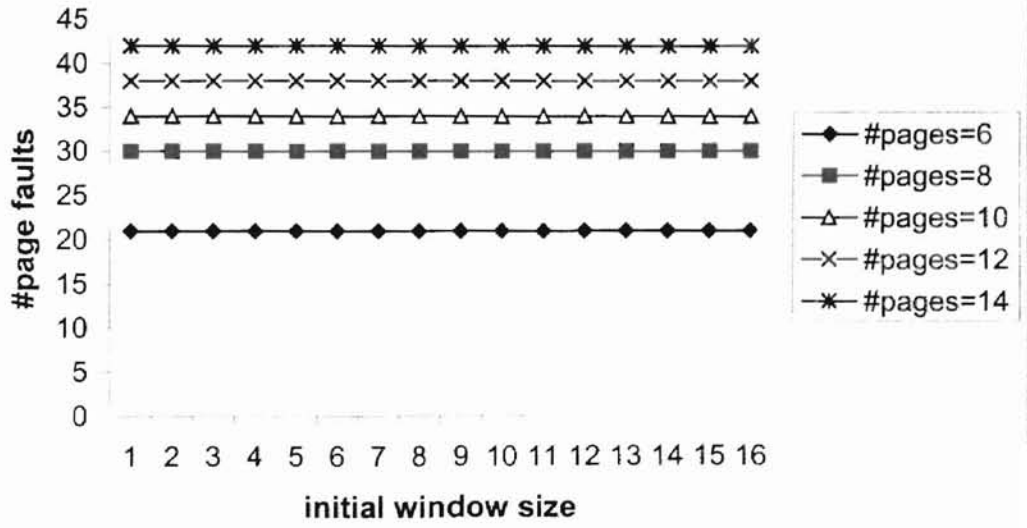


Figure 7: ELRU With Different #Distinct Pages In PLM - 2



Figures [6] and [7] are simulations to test the effect of initial window size on the number of page faults under the independent reference model and pure loops model. The number of frames is randomly set to 5 in this simulation. We can see that for the independent reference model, the different values of initial locality window size cause variation of number of page faults under the fixed number of frames and reference string whereas the number of page faults remains the same for the pure loops model. Since the length of reference strings are the same, when the number of pages is smaller, we can assume that the reference string generally exhibits better spatial and temporal locality, hence, achieves less number of page faults. Figure 7 also gives a verification of Claim 7.

Figure 8: ELRU With Different # Frames In IRM

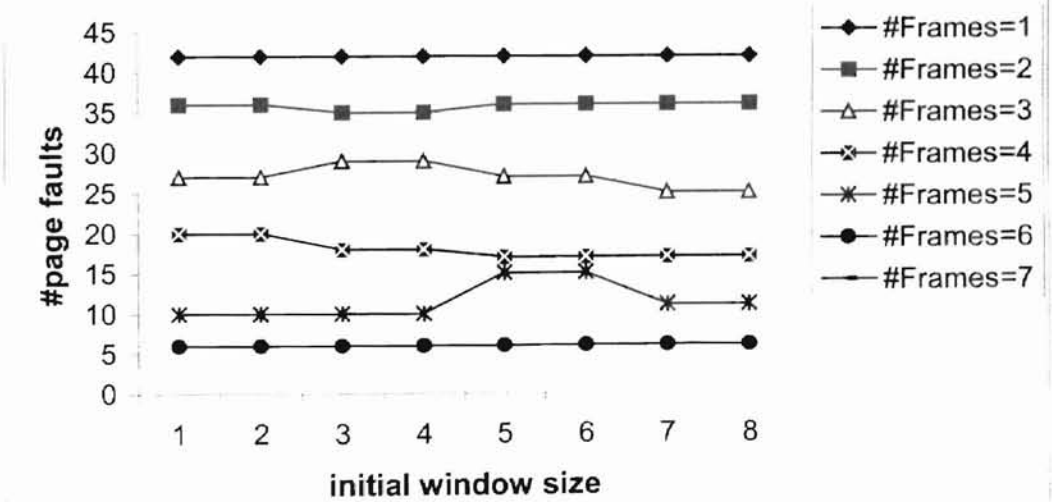


Figure [8] is to study the effect of number of allocated frames and initial locality window size on the number of page faults under the independent reference model. The number of pages is randomly set to 6 in this simulation. Two facts are shown in this figure. One is that the more the number of frames, the less the number of page faults no matter what the initial locality window size is. The other is that when the number of frames is greater than 1 and less than the number of pages, initial locality window size does affect the number of page faults under this model. Two lines with #Frames of 6 and 7 coincide since they have number of frames no less than the number of pages and hence the numbers of page faults are both the number of pages. Sometimes larger initial window size cause more number of page faults. The reason may be because that spatial locality of page w means that pages in the range of $(w-i, w+i)$ for some small i are likely referenced next. If the window size is too large, i.e., i is too large, pages in the range of $(w-i, w+i)$ may be far away from the current page.

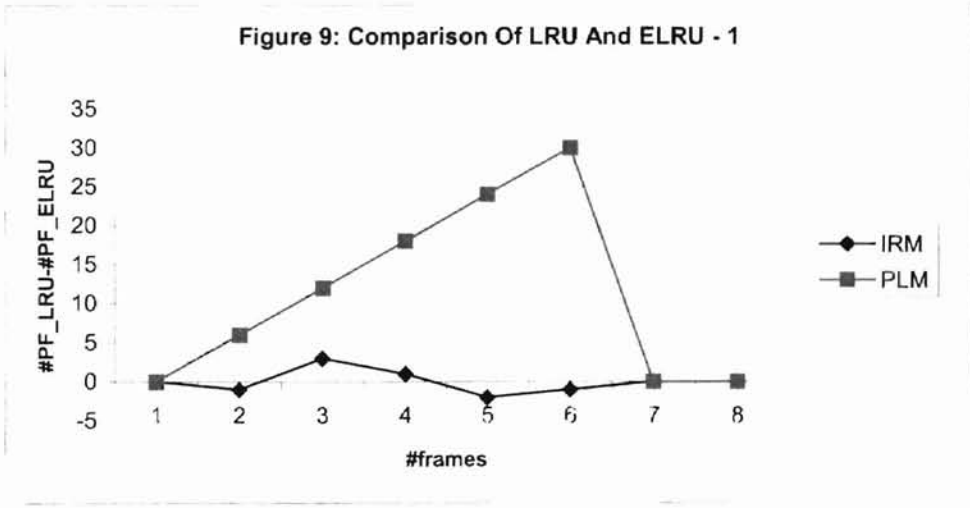


Figure 10: Comparison Of LRU And ELRU - 2

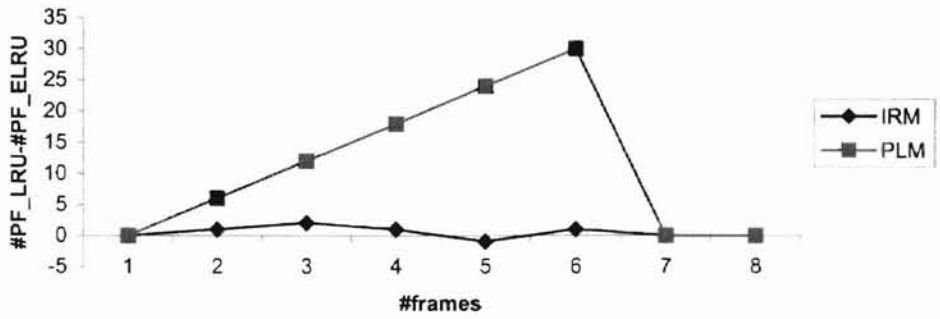


Figure 11: Comparison Of LRU And ELRU - 3

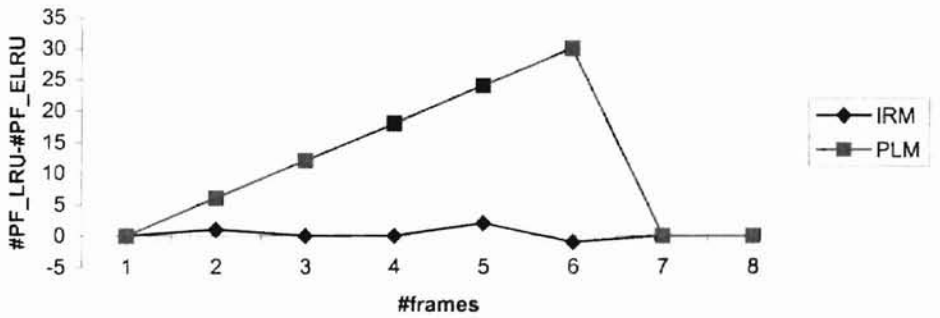
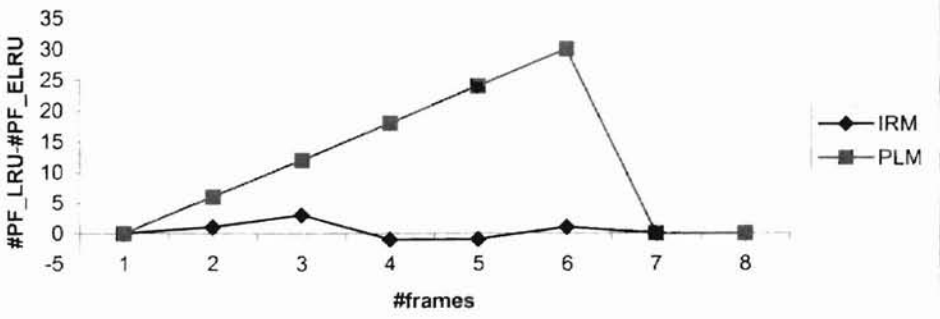
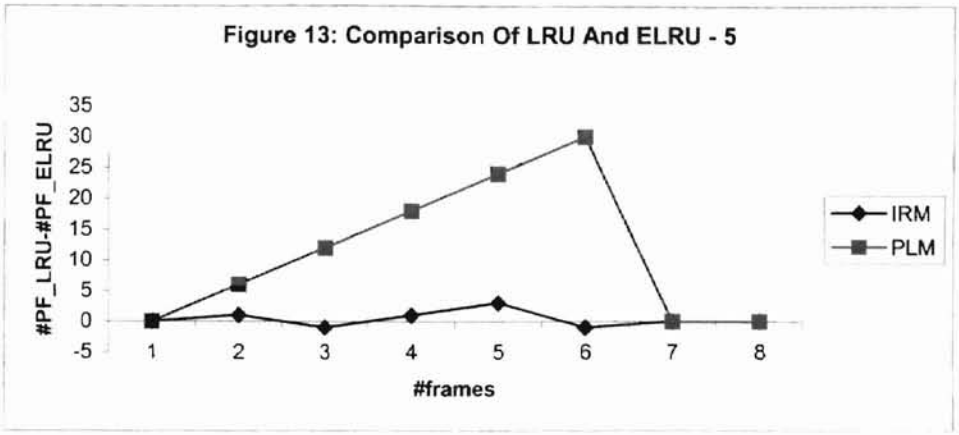


Figure 12: Comparison Of LRU And ELRU - 4





Figures [9] – [13] are to compare performances of the LRU and the proposed algorithm for five different random reference strings under the independent reference model and pure loops model. The x-axis is the number of frames and the y-axis is the difference of the number of page faults in LRU and the proposed algorithm, i.e., #PF_LRU-#PF_ELRU. The number of pages is set to 7, the length of reference string is set to 50, and the initial locality window size is set to 3 randomly. In the pure loops model, the ELRU always performs better than LRU and the more the number of frames, the better the ELRU than LRU. On the other hand, in the independent reference model, LRU and ELRU are comparable. After the number of frames is equal to or more than 7, the number of pages, the difference is always 0 since both LRU and ELRU have the number of page faults equal to the number of pages. By studying five corresponding reference strings listed below, it is shown that the better the spatial locality exhibited, the less the number of page faults in the proposed algorithm. This confirms the performance analysis with respect to the number of page faults conducted in section 5.4.

The reference string used in Figures[9]-[13] for PLM is 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 in this simulation.

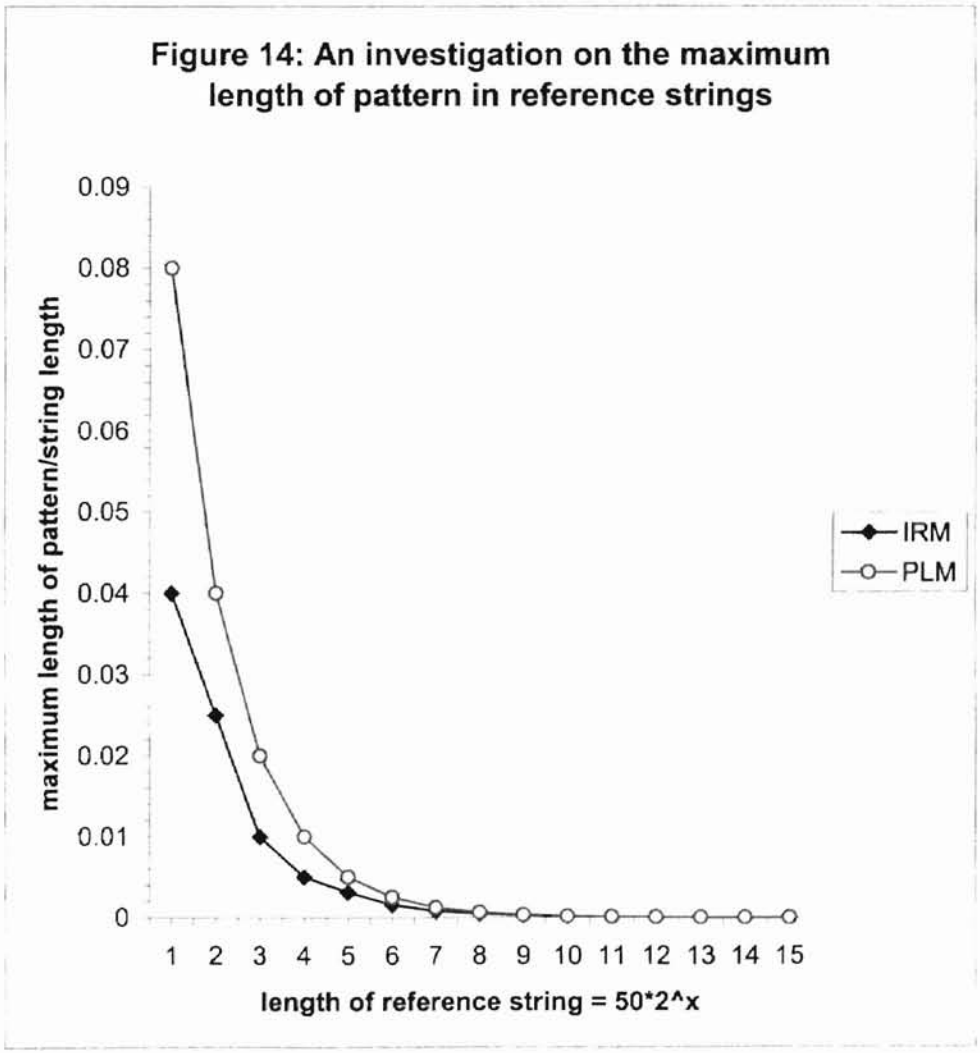
The reference strings used in Figures [9]-[13] for IRM in this simulation are listed in Table IV.

Table IV: Reference Strings Used In Figures [9]-[13]

Fig.	Reference string in IRM in Figure
8	0 3 0 1 3 4 6 0 6 5 2 0 4 5 5 6 2 5 3 2 0 6 6 6 6 0 0 4 1 5 4 4 5 0 5 1 5 4 1 6 0 4 6 2 6 1 3 0 2 1
9	6 4 0 6 2 0 1 1 2 3 2 3 0 6 5 2 3 4 4 5 5 1 6 0 1 4 1 0 2 2 3 4 4 1 4 2 3 2 4 3 0 3 4 6 6 5 4 5 4 3
10	6 3 1 3 6 6 2 6 5 5 2 6 6 5 5 2 0 0 1 0 0 3 6 0 6 5 0 1 1 1 6 6 0 2 1 1 2 4 4 1 5 6 4 1 6 2 2 0 1 4
11	3 4 3 5 1 3 2 2 0 1 0 3 4 0 2 3 4 0 5 4 6 0 4 4 0 5 1 6 1 2 0 4 4 6 5 2 4 3 5 0 3 0 2 3 1 5 1 5 1 2
12	5 0 2 0 4 6 2 1 4 1 0 3 1 0 2 3 2 6 3 1 5 2 5 3 6 5 6 0 6 5 5 4 6 6 3 1 4 3 1 1 2 5 5 6 4 1 3 2 1 1

Figure [14] is a simulation to study the maximum length of pattern (defined in section 5.4) in reference strings under both the independent reference model and the pure loops model because the existence of pattern leads to some extra overhead in time and space in the proposed algorithm. In this simulation, randomly, the number of allocated frames is set to 5, the number of distinct pages is set to 8, and the initial window size is set to 3. The x-axis is used to compute the length of the reference string, which is 50×2^x where x is the x-axis. So, in this simulation, the length of the reference string varies from $50 \times 2^1 = 100$ to $50 \times 2^{15} = 1638400$. The y-axis is the quotient of the maximum length of pattern in the reference string and the length of the reference string. It is shown that in

both models the maximum quotient is 0.8 =the number of distinct pages over the shortest reference string length. Moreover, it approaches 0 rapidly as the reference string length increases since the maximum length of pattern only has slight changes whereas the length of reference string increases exponentially. Therefore, the overhead caused by the usage of pattern in the proposed algorithm can be negligible on the average under both the independent reference model and the pure loops model.



CHAPTER 5

CONCLUSIONS

The LRU algorithm is known to be a good approximation to optimal page replacement algorithm yielding small number of page faults. However, it has a problem that the selection of a victim page is based only on the time of last reference, with consequent harm to result in high memory miss ratio for the reference string with good temporal and spatial locality and a large number of productive pages. The proposed extended LRU algorithm introduces adaptive locality window size and locality function into the conventional LRU algorithm in an attempt to exploit both spatial and temporal locality of reference strings during the program run time. It is formally proven that the proposed algorithm is a stack algorithm, hence it will not suffer from Belady's anomaly, which enables it to be predictable and thereby to be improved. Under the assumption of pure loops model, the proposed algorithm must perform not worse than LRU with respect to minimizing the number of page faults. The more the number of allocated frames within the number of total distinct pages, the better the proposed algorithm than LRU. Under the assumption of independent reference model, the proposed algorithm and LRU are comparable in terms of the number of page faults. Moreover, the proposed algorithm causes some extra overhead in time and space due to the implementation of adaptation of locality window size and locality function. However, this overhead is negligible and the proposed algorithm is comparable in efficiency to the LRU algorithm on the average.

REFERENCES

[Abraham and Sugumar 93] S.G. Abraham and R.A. Sugumar, "Efficient Simulation of Caches Under Optimal Replacement with Applications to Miss Characterization", *Proc. ACM Int'l Conf. Measurement and Modeling of Computer Systems*, Santa Clara, CA, USA, May 1993.

[Agarwal and Pudar 93] A. Agarwal and S. D. Pudar, " Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches", *Proc. 20th Int'l Symp. Computer Architecture*, ACM Inc., pp.179-190, San Diego, Calif., May 1993.

[Belady 66] L.A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer", *IBM Systems J.*, Vol. 5, No. 2, pp. 78-101. 1966.

[Burger et al. 96] D. Burger, A. Kagi and J.R. Goodman, "Memory Bandwidth Limitations of Future Microprocessors", *Proc. 23rd ACM Int'l Symp. Computer Architecture*, ACM Inc., Philadelphia, May 1996.

[Carr 84] R.W. Carr, *Virtual Memory Management*, UMI Research Press, Ann Arbor, MI, 1984.

[Chrobak and Noga 98] M. Chrobak and J. Noga, "LRU is better than FIFO", *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM Inc., pp. 78-81, 1998.

[Denning 70] P.J. Denning, "Virtual Memory", *ACM Computing Surveys*, Vol. 2, No.3, pp. 153-189, September 1970.

[Denning 72] P.J. Denning, "On Modeling Program Behavior", *AFIPS, Conf. Proc. 40*, ACM Inc., pp. 937-944, 1972.

[Hyde 88] R. L. Hyde, "Overview of Memory Management", *Byte*; Peterborough, N.H., Vol. 13, No. 4, pp. 219-225, 1988.

[Jain 90] R. Jain, "Characteristics of Destination Address Locality in Computer Networks: A Comparison of Caching Schemes", *Computer Networks And ISDN Systems*, Vol. 18, No. 4, pp. 243-254, Amsterdam, May 1990.

[Jouppi 90] N.P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small, Fully-Associative Cache and Prefetch Buffers", *Proc. 17th ACM Int'l Symp. Computer Architecture*, ACM Inc., pp. 364-373, May 1990.

[Martin et al. 90] T. P. Martin, I. A. Macleod, J. I. Russell, K. Leese and B. Foster, "A Case Study of Caching Strategies for a Distributed Full Text Retrieval system", *Information Processing & Managemant*, Vol. 26, No. 2, pp. 227-247, Oxford, 1990.

[Maruyama 75] K. Maruyama, "MLRU Page Replacement Algorithm in Terms of the Reference Matrix", *IBM Tech. Disc. Bull.*, Vol. 17, No. 10, pp. 3101-3103, March 1975.

[Mattson et al. 70] R.L. Mattson, J. Gecsei, D.R. Slutz, and L.L. Traiger, "Evaluation Techniques for Storage Hierarchies", *IBM Systems J.*, Vol. 9, No. 2, pp. 78-117, 1970.

[McFarling 92] S. McFarling, "Cache Replacement with Dynamic Exclusion", *Proc. 19th Ann. Int'l Symp. Computer Architecture*, ACM Inc., pp. 191-200, Gold Coast, Australia, 19-21 May 1992, also in *Computer Architecture News*, Vol. 20, No. 2, May 1992.

[Moote 89] R. Moote, "Virtual Memory: The Next Generation", *Byte*; Peterborough, N.H., Vol. 14, No. 12, pp. 342-350, 1989.

[O'Neil et al. 93] E.J. O'Neil, P.E. O'Neil and G. Weikum, "The LRU-K Page Replacement Algorithm for Database Disk Buffering", *In Proceedings of the 1993 SIGMOD Int'l Conf. on Management of Data*, (Washington, D.C., May 26-28), ACM, pp. 297-306, New York, 1993.

[O'Neil et al. 99] E.J. O'Neil, P.E. O'Neil and G. Weikum, "An Optimality Proof of the LRU-K Page Replacement Algorithm", *Journal of the Association for Computing Machinery*, Vol. 46, No. 1, pp. 92-112, New York, Jan. 1999.

[Schneider 83] R.J. Schneider, *A Demand Driven Data Flow Environment for a Locality Study*, Thesis, Oklahoma State University, 1983.

[Seznec 93] A. Seznec, "A Case for Two-Way Skewed-Associative Caches", *Proc. 20th Int'l Symp. Computer Architecture*, ACM Inc., pp. 169-178, San Diego, Calif., May 1993.

[Silberschatz and Galvin 97] A. Silberschatz and P.B. Galvin, *Operating System Concepts*, Fifth Edition, Addison-Wesley Longman, Inc., Reading, MA, 1997.

[Spirn 77] J.R. Spirn, *Program Behavior: Models and Measurement*, Ed. Peter J. Denning, New York: Elsevier North-Holland, Inc., pp. 45-50, 1977.

[Tanenbaum 92] A. S. Tanenbaum, *Modern Operating Systems*, Prentice Hall, Inc., 1992.

[Temam 99] O. Temam, "An Algorithm for Optimally Exploiting Spatial and Temporal Locality in Upper Memory Levels", *IEEE Transactions on Computers*, Vol. 48, No. 2, pp. 150-158, 1999

[Thoreson and Long 89] S.A. Thoreson and A.N. Long, "Locality, a Memory Hierarchy, and Program Restructuring in a Dataflow Environment", *The Journal of Systems and Software*, Vol. 9, No. 4, pp. 245-252, New York, May 1989.

[Trivedi 76] K. S. Trivedi, "Prepaging and Applications to Array Algorithms", *IEEE Transactions on Computers*, Vol. C-25, No. 9, pp. 915-921, September 1976.

[Van Den Berg and Gandolfi 92] J. Van Den Berg and A. Gandolfi, "LRU is Better Than FIFO Under the Independent Reference Model", *Journal of Applied Probability*, Vol. 29, No. 1, pp. 239-243, Sheffield, Mar. 1992.

APPENDIX A: THE PROOF OF CLAIM 2

Claim 2: For any reference string in PLM_kn, if memory size is m and the locality window size set in the proposed algorithm is w, then the total number of page faults is

$$NumPF = \begin{cases} k, & \text{if } k \le m, \\ k + N_{PF}(n-1), & \text{otherwise,} \end{cases}$$

where

$$N_{PF} = \begin{cases} k - (H_f + H_m + H_l), & \text{if } m + w - k > 0, \\ k, & \text{otherwise.} \end{cases}$$

$$H_f = \min(m + w - k, m - 1),$$

$$H_m = \sum_{i=1}^{\left\lfloor \frac{k}{w} \right\rfloor - 1} H,$$

$$H = \begin{cases} m + w - k, & \text{if } i \le \left\lfloor \frac{m-1}{w} \right\rfloor, \\ & \text{and} \\ 0, & \text{otherwise,} \end{cases}$$

$$H_l = \begin{cases} m - \left\lfloor \frac{k}{w} \right\rfloor w, & \text{if } \left\lfloor \frac{k}{w} \right\rfloor = \left\lfloor \frac{m-1}{w} \right\rfloor \text{ and } w < k, \\ 0, & \text{otherwise.} \end{cases}$$

Proof: If $k \le m$, then the number of distinct pages is no more than the memory capacity. This is demand paging and memory is empty initially, so there are k page faults for storing k pages in memory. After the first reference to page k, every page is in memory, and there are no more page faults at all. Therefore, there are totally k page faults in this case.

The following discussion is based on the condition of $k > m$.

N_{PF} is actually the number of page faults in one iteration except for the first one.

Without loss of generality, we analyze the second loop here.

For the sake of simplicity, the following numbers associated with page mean indices of pages.

After the first reference to page k, the memory content is pages $\{k, k-1, ..., k-m+1\}$.

At reference to page 1, the locality window contains pages $\{1,2,..., w\}$, so the number of common pages in memory and locality window is

$$N_o = \begin{cases} 0 & \text{if } w < k \text{ and } m + w \le k, \\ m + w - k & \text{if } w < k \text{ and } m + w > k, \\ m & \text{if } w \ge k. \end{cases}$$

Case 1: $w \ge k$

Then every page in memory now is in the locality window. So the victim page is page $k-m+1$ and pages $\{k-m+2,...,k\}$ are all in the window.

$k - m + 1 \ge 2 \Rightarrow k - m + 2 >= 3$.

Before the reference to page $k-m+2$, pages $\{k-m+2,...,k\}$ are always both in the window and in memory and there is another page in memory to serve as a victim, so references to these pages are all hits and the rest cause page faults. Therefore, the number of hits is $m-1$.

Case 2: $w<k$ and $m+w\leq k$

Then $w\leq k-m<k-m+1$, so there is no overlap between memory and the locality window at reference to page 1. Therefore, the proposed algorithm selects the same victim page as LRU, which is the least recently used page $k-m+1$. There is one page fault occurring. At reference to page 2, page 1 is added to memory and page $k-m+1$ is removed. Even page $w+1$ is in the window now, since $w+1<k-m+2$ and page 1 has not resided in the window any more, there is still no overlap existing between memory and the window and one page fault exists. Keep doing this, we can see that at any reference, window and memory are disjoint and every reference is a miss. So the number of hits is 0.

Case 3: $w<k$ and $m+w>k$

$k>m$ implies that $k-m+1>1$, and $w<k$, so the overlap of memory and the window at reference to page 1 must be pages $\{k-m+1,k-m+2,...,w\}$ and the victim page is page $w+1$. Before the reference to page $k-m+1$, every reference is a miss and since from page i to page $i+1$, even window content changes from $\{i,i+1,...,i+w-1\}$ to $\{i+1,i+2,...,i+w-1,i+w\}$, if $i+w$ is in memory at i, it will be removed from memory since it is the one outside the window with the lowest time stamp. Therefore, the overlap

from page 1 to page $k-m$ are always the same, which is pages $\{k-m+1, k-m+2, ..., w\}$ and references to these pages are hits. So the number of hits from page 1 to page w $= w-(k-m+1)+1 = w+m-k$.

At reference to page $nw+1$ for $n \geq 1$, locality window content is pages $\{nw+1, nw+2, ..., nw+w\}$ and memory content is

$$\begin{cases} nw-i \text{ if } nw-i>0, \\ \qquad\qquad \text{for } 0 \leq i \leq m-1, \\ nw-i+k \text{ otherwise,} \end{cases}$$

If $m-1 < nw$, then $nw-i$ will always be greater than 0. So memory content will be pages $\{nw-1, nw-2, ..., nw-m+1\}$, and there will be no overlap between memory and the locality window, hence there is no hits.

The following justification is based on the condition of $m-1 \geq nw$, i.e., $n \leq \frac{m-1}{w}$.

There are two possibilities for the relation between k, w, and n.

(1) $k \geq (n+1)w$

$nw-i < nw+1$ for $i \geq 0$, so $i \geq nw$ is the necessary condition for that pages $nw-i+k$ are in the window.-----(a)

if $1 \leq k-i \leq w$, then $nw+1 \leq nw+k-i \leq nw+w$, hence page $nw-i+k$ is in the window. In other words, when $k-w \leq i \leq k-1$, pages $nw-i+k$ are in the window.-----(b)

$k \geq (n+1)w$ implies that $k-w \geq nw$ and $k > m$ implies that $k-1 > m-1$. -----(c)

By (a), (b), (c), and $0 \le i \le m-1$, the sufficient and necessary condition for pages $nw-i+k$ residing in the window is $k-w \le i \le m-1$. That is, the number of common pages between memory and the window is $m-1-(k-w)+1 = m+w-k$. By the similar discussion at the beginning of case 3, we have that from page nw to page $(n+1)w$, before the first element in the overlap, every reference is a miss and the overlap remains the same. So the number of hits from page nw to page $(n+1)w$ is $m+w-k$.

(2) $nw < k < (n+1)w$

Even the locality window contains pages $\{nw+1, nw+2, ..., nw+w\}$, since $k < nw+w$, $nw-i+k$ may not be greater than k. $nw+1 \le nw+k-i \le k \Rightarrow 1 \le k-i \le k-nw$. So when $nw \le i \le k-1$, pages $nw-i+k$ are in window. Since $0 \le i \le m-1$ and $k-1 > m-1$, the sufficient and necessary condition for pages $nw-i+k$ in the window is $nw \le i \le m-1$. Therefore, the number of hits $= m-1-nw+1 = m-nw$ from page $nw+1$ to page k.

In a word, let $n = \left\lfloor \frac{k}{w} \right\rfloor$, H_f stand for the number of hits from page 1 to page w, H the number of hits from page $nw+1$ to page $nw+w$ for $1 \le n \le \left\lfloor \frac{k}{w} \right\rfloor - 1$, and H_l the number of hits from page $nw+w+1$ to page k, during one iteration(page 1 to page k), we get

(i) If $w \ge k$, then $m+w-k > 0$ and the number of hits is $m-1$, i.e., $H_f = m-1$.

(ii) If $w < k$ and $m+w-k \le 0$, then the number of hits $= 0$. That is,

$H_f = H = H_l = 0$.

(iii) If $w < k$ and $m + w - k > 0$, then $H_f = m + w - k$, $H = m + w - k$ if

$n \le \left\lfloor \frac{m-1}{w} \right\rfloor$ and 0 otherwise, and $H_l = m - nw$ if $n \le \left\lfloor \frac{m-1}{w} \right\rfloor$ and 0 otherwise.

$w \ge k > m \Rightarrow m + w - k \ge m > m - 1$,

$w < k \Rightarrow m + w - k < m \Rightarrow m + w - k \le m - 1$,

$$\text{so } \min(m + w - k, m - 1) = \begin{cases} m - 1, \text{ if } w \ge k, \\ \\ m + w - k \text{ if } w < k. \end{cases}$$

so $H_f = \min(m + w - k, m - 1)$ if $m + w - k > 0$.

As a result, if $k > m$, then the number of page faults during one iteration is

$$N_{PF} = \begin{cases} k - (H_f + H_m + H_l), & \text{if } m + w - k > 0, \\ \\ k, & \text{otherwise.} \end{cases}$$

where

$$H_f = \min(m + w - k, m - 1),$$

$$H_m = \sum_{i=1}^{\left\lfloor \frac{k}{w} \right\rfloor - 1} H,$$

$$H = \begin{cases} m + w - k, & \text{if } i \le \left\lfloor \frac{m-1}{w} \right\rfloor, \\ \\ 0, & \text{otherwise,} \end{cases}$$

and

$$H_i = \begin{cases} m - \left\lfloor \frac{k}{w} \right\rfloor w, & \text{if } \left\lfloor \frac{k}{w} \right\rfloor = \left\lfloor \frac{m-1}{w} \right\rfloor \text{ and } w < k, \\ 0, & \text{otherwise.} \end{cases}$$

Since except for the first loop, the number of page faults are the same. Hence, the total number of page faults is $k + N_{PF}(N-1)$. The claim is proved.

APPENDIX B: THE PROOF OF CLAIM 3

Claim 3: For any reference string in PLM_kn, let memory size be m, then the page-fault rate in LRU is

$$PFRate = \begin{cases} 1/N, \text{ if } k \le m, \\ \\ 100\%, \text{ otherwise.} \end{cases}$$

Proof: Let a reference string in PLM_kn is $x_1, x_2, ..., x_k, x_1, x_2, ...x_k, ..., x_1, x_2, ..., x_k$,

Case 1: if $k \le m$, obviously, the total number of page faults is k occurring during the first k distinct page references. After that, every reference is a hit since every page is in memory. So the page-fault rate is $k/kN = 1/N$.

Case 2: if $k > m$,

Since each page in one iteration is distinct and initially memory is empty, first k references will cause k page faults. After the first k references, since $k > m$ and all k pages are distinct, at every reference x_m where $1 \le m \le k$, the memory content is

$$\begin{cases} x_{m-i} \text{ if } m-i > 0, \\ \\ \qquad\qquad \text{for } 1 \le i \le m, \\ \\ x_{k+m-i} \text{ otherwise,} \end{cases}$$

$k > m$ and $i \le m \Rightarrow i < k \Rightarrow k - i + m > m$, $1 \le i \Rightarrow m - i < m$, so x_m is not in memory, hence, there is a page fault. Therefore, every reference causes a page fault. So the page-fault rate is 100%.

APPENDIX C: THE PROOF OF CLAIM 4

Claim 4: For any reference string in PLM_kn, if the locality window size w set in the proposed algorithm is greater than or equal to $k-1$, then the number of page faults is independent of w.

Proof: Without loss of generality, we assume that $m > 1$.

By Claim 2,

Case 1: If $k \le m$, then $N_{PF} = k$, so the claim is true.

Case 2: $k > m$, then $k > 1$.

$w \ge k-1 \Rightarrow m+w-k \ge m-1 > 0$, so $N_{PF} = k-(H_f + H_m + H_l)$.

$H_f = \min(m+w-k, m-1) = m-1,$

$\frac{k}{w} \le \frac{k}{k-1} = 1 \Rightarrow \left\lfloor \frac{k}{w} \right\rfloor - 1 \le 0 \Rightarrow H_m = 0,$

If $w = k-1$,

then $\left\lfloor \frac{k}{w} \right\rfloor = 1$. And

$$k > m \Rightarrow w = k-1 > m-1 \Rightarrow \left\lfloor \frac{m-1}{w} \right\rfloor = 0 \Rightarrow \left\lfloor \frac{k}{w} \right\rfloor \ne \left\lfloor \frac{m-1}{w} \right\rfloor .$$

So $H_l = 0$.

Else, $w > k-1 \Rightarrow w \ge k \Rightarrow H_l = 0$.

Therefore, $N_{PF} = k-(m-1) = k-m+1$, which is independent of w.

That is, the claim is true.

APPENDIX D: THE PROOF OF CLAIM 5

Claim 5: N_{PF} given in Claim 2 is a non-increasing function of the locality window size w.

Proof: Without loss of generality, we assume that $k > m$ and $m \geq 2$.

In order to prove claim 5, it suffices to prove that $N_{PF}(w) \geq N_{PF}(w+1)$ for $w \geq 1$.

There are 3 cases for the relationships of w, $w+1$, and k:

Case 1: $w \geq k$ and $w+1 \geq k$.

Case 2: $w < k$ and $w+1 = k$.

Case 3: $w+1 < k$ and $w < k$.

Case 1: Since $w \geq k$ and $w+1 \geq k$, by the proof of claim 2, we know that $N_{PF}(w) = N_{PF}(w+1) = k-(m-1)$. So $N_{PF}(w) \geq N_{PF}(w+1)$.

Case 2: Similarly, $N_{PF}(w+1) = k-(m-1)$.

$w+1 = k \Rightarrow w = k-1 \Rightarrow m+w-k = m-1 > 0$ and $\left\lfloor \frac{k}{w} \right\rfloor = 1$, so by claim 2,

$H_f = \min(m-1, m-1) = m-1$.

$\left\lfloor \frac{k}{w} \right\rfloor - 1 = 0 \Rightarrow H_m = 0$.

$k > m \Rightarrow w = k-1 > m-1 \Rightarrow \frac{m-1}{w} < 1 \Rightarrow \left\lfloor \frac{k}{w} \right\rfloor > \left\lfloor \frac{m-1}{w} \right\rfloor \Rightarrow H_l = 0$.

So $N_{PF}(w) = k-(H_f + H_m + H_l) = k-(m-1)$. Therefore, $N_{PF}(w) \geq N_{PF}(w+1)$.

Case 3: There are 3 subcases existing for this case:

Subcase 1: $w < k-m$ and $w+1 \leq k-m$.

Subcase 2: $w = k - m$ and $w+1 = k - m + 1$.

Subcase 3: $k > w > k - m$ and $k > w+1 > k - m$.

Subcase 1: By claim 2, $w+1 \le k - m \Rightarrow m + (w+1) - k \le 0 \Rightarrow N_{PF}(w+1) = k$

$w < k - m \Rightarrow m + w - k < 0 \Rightarrow N_{PF}(w) = k$. So $N_{PF}(w) \ge N_{PF}(w+1)$.

Subcase 2: For the same reason, $N_{PF}(w) = k$.

$w+1 = k - m + 1 \Rightarrow m + (w+1) - k = 1 \Rightarrow H_f = \min(m + (w+1) - k, m-1) = 1$.

So $N_{PF}(w+1) = k - (H_f + H_m + H_l) < k$. That is, $N_{PF}(w) \ge N_{PF}(w+1)$.

Subcase 3: $m + w - k > 0$ and $m + w + 1 - k > 0$

$w < k \Rightarrow m + w - k < m \Rightarrow m + w - k \le m - 1 \Rightarrow H_f^w = m + w - k$.

$w+1 < k \Rightarrow m + w + 1 - k \le m - 1 \Rightarrow H_f^{w+1} = m + w + 1 - k$.

If $\left\lfloor \frac{k}{w} \right\rfloor = \left\lfloor \frac{k}{w+1} \right\rfloor \ge 1$, then

$$w > k - m \Rightarrow m > k - w \Rightarrow m - 1 \ge k - w \Rightarrow \frac{m-1}{w} \ge \frac{k-w}{w} = \frac{k}{w} - 1 \text{-----(1)}$$

Similarly, $\frac{m-1}{w+1} \ge \frac{k}{w+1} - 1$-----(2)

$m < k \Rightarrow m - 1 < k \Rightarrow \frac{m-1}{w} \le \frac{k}{w}$-----(3) and

$\frac{m-1}{w+1} \le \frac{k}{w+1}$-----(4)

From (1), (2), (3), and (4), we have that either $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor$ or $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor - 1$,

and either $\left\lfloor \frac{m-1}{w+1} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor$ or $\left\lfloor \frac{m-1}{w+1} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor - 1$.

So either $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{m-1}{w+1} \right\rfloor$ or $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{m-1}{w+1} \right\rfloor + 1$.

If $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{m-1}{w+1} \right\rfloor$, then

$H_m^{w+1} - H_m^w = t(m+w+1-k) - t(m+w-k) = t \geq 0$ where

$t = \left\lfloor \frac{k}{w} \right\rfloor - 1$ if $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor$ or $t = \left\lfloor \frac{m-1}{w} \right\rfloor$ if $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor - 1$,

$H_l^{w+1} - H_l^w = -\left\lfloor \frac{k}{w+1} \right\rfloor$ if $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor$ or $H_l^{w+1} - H_l^w = 0$ if $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor - 1$,

and $H_f^{w+1} - H_f^w = (m+w+1-k) - (m+w-k) = 1$.

So $N_{PF}(w) - N_{PF}(w+1) = H_f^{w+1} + H_m^{w+1} + H_l^{w+1} - (H_f^w + H_m^w + H_i^w) = 0$ if

$\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor$ and $N_{PF}(w) - N_{PF}(w+1) = \left\lfloor \frac{m-1}{w} \right\rfloor + 1 > 0$ if $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor - 1$.

That is, $N_{PF}(w) \geq N_{PF}(w+1)$ if $\left\lfloor \frac{k}{w} \right\rfloor = \left\lfloor \frac{k}{w+1} \right\rfloor \geq 1$ and $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{m-1}{w+1} \right\rfloor$.

If $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{m-1}{w+1} \right\rfloor + 1$, then

$\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor$ and $\left\lfloor \frac{m-1}{w+1} \right\rfloor = \left\lfloor \frac{k}{w} \right\rfloor - 1$.

$H_m^w = (\left\lfloor \frac{k}{w} \right\rfloor - 1)(m+w-k)$, $H_m^{w+1} = (\left\lfloor \frac{k}{w+1} \right\rfloor - 1)(m+w+1-k)$,

$H_l^w = m - \left\lfloor \frac{k}{w} \right\rfloor w$, and $H_l^{w+1} = 0$.

Since $\left\lfloor \frac{k}{w} \right\rfloor = \left\lfloor \frac{k}{w+1} \right\rfloor \geq 1$, let $p = \left\lfloor \frac{k}{w} \right\rfloor = \left\lfloor \frac{k}{w+1} \right\rfloor$. Then

$H_f^w + H_m^w + H_l^w = (m+w-k) + pm + pw - pk - m - w + k + m - pw = pm - pk + m$ and

$H_f^{w+1} + H_m^{w+1} + H_l^{w+1} = pm + pw + p - pk$,

So, $N_{PF}(w) - N_{PF}(w+1) = H_f^{w+1} + H_m^{w+1} + H_l^{w+1} - (H_f^w + H_m^w + H_l^w) = pw + p - m$.

If $pw + p < m$, then $\left\lfloor \frac{k}{w} \right\rfloor (w+1) \le m-1 \Rightarrow \frac{m-1}{w+1} \ge \left\lfloor \frac{k}{w} \right\rfloor \Rightarrow \left\lfloor \frac{m-1}{w+1} \right\rfloor \ge \left\lfloor \frac{k}{w} \right\rfloor > \left\lfloor \frac{k}{w+1} \right\rfloor - 1$, a contradiction. So $pw + p \ge m$. That is, $N_{PF}(w) - N_{PF}(w+1) = pw + p - m \ge 0$ if $\left\lfloor \frac{k}{w} \right\rfloor = \left\lfloor \frac{k}{w+1} \right\rfloor \ge 1$ and $\left\lfloor \frac{m-1}{w} \right\rfloor = \left\lfloor \frac{m-1}{w+1} \right\rfloor + 1$.

Therefore, $N_{PF}(w) \ge N_{PF}(w+1)$ when $\left\lfloor \frac{k}{w} \right\rfloor = \left\lfloor \frac{k}{w+1} \right\rfloor \ge 1$.

By the similar discussion, we can get that $N_{PF}(w) \ge N_{PF}(w+1)$ when $\left\lfloor \frac{k}{w} \right\rfloor > \left\lfloor \frac{k}{w+1} \right\rfloor$.

So $N_{PF}(w) \ge N_{PF}(w+1)$ for $w \ge 1$. The claim is proved.

APPENDIX E: THE PROOF OF CLAIM 7

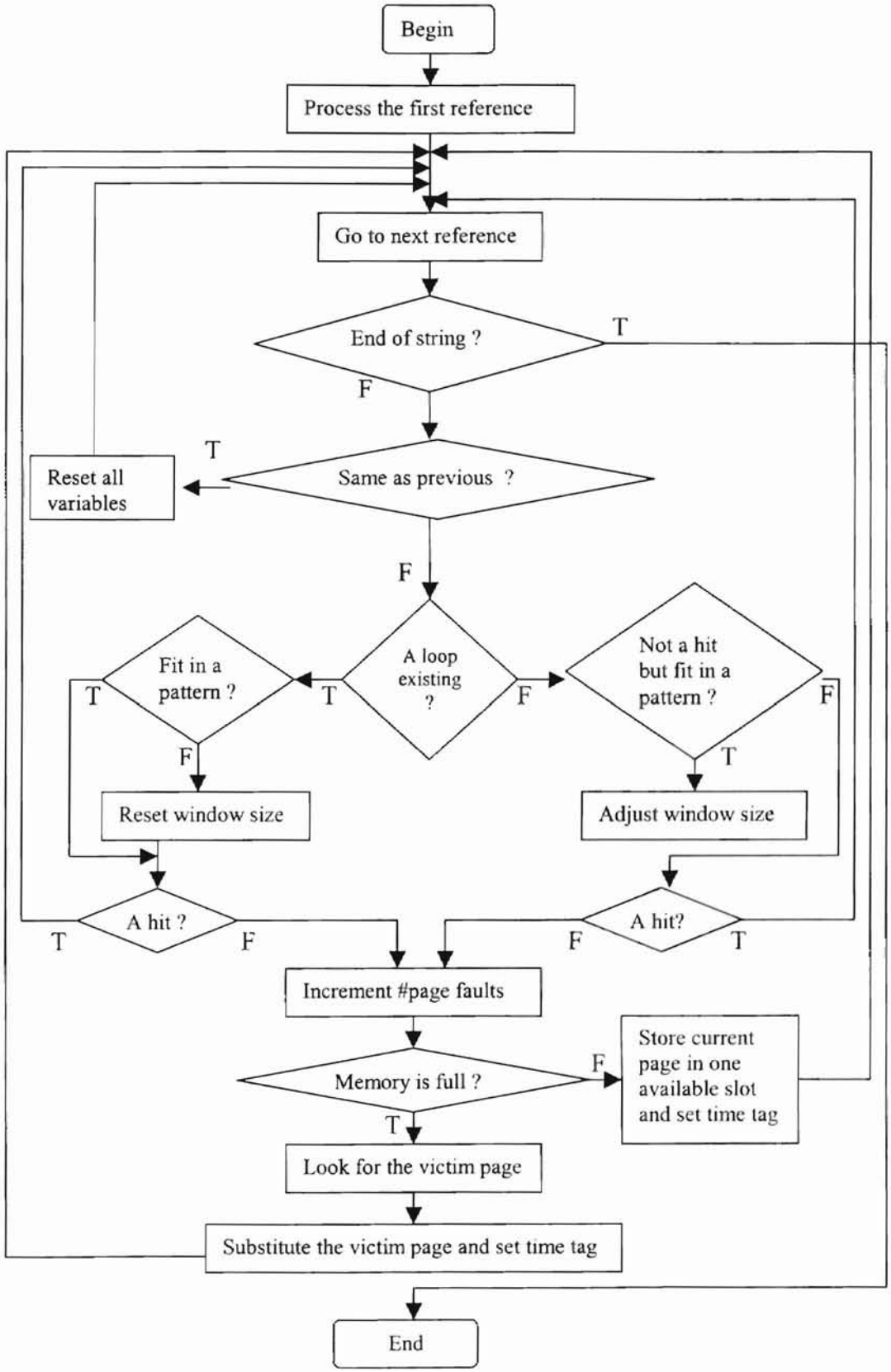
Claim 7: Initial locality window size has no effect on the number of page faults for any reference string in PLM_kn in the proposed algorithm.

Proof: Let the reference string be $x_1, x_2, ..., x_k, x_1, ..., x_k, ...$, initial locality window size be w, and memory size be m.

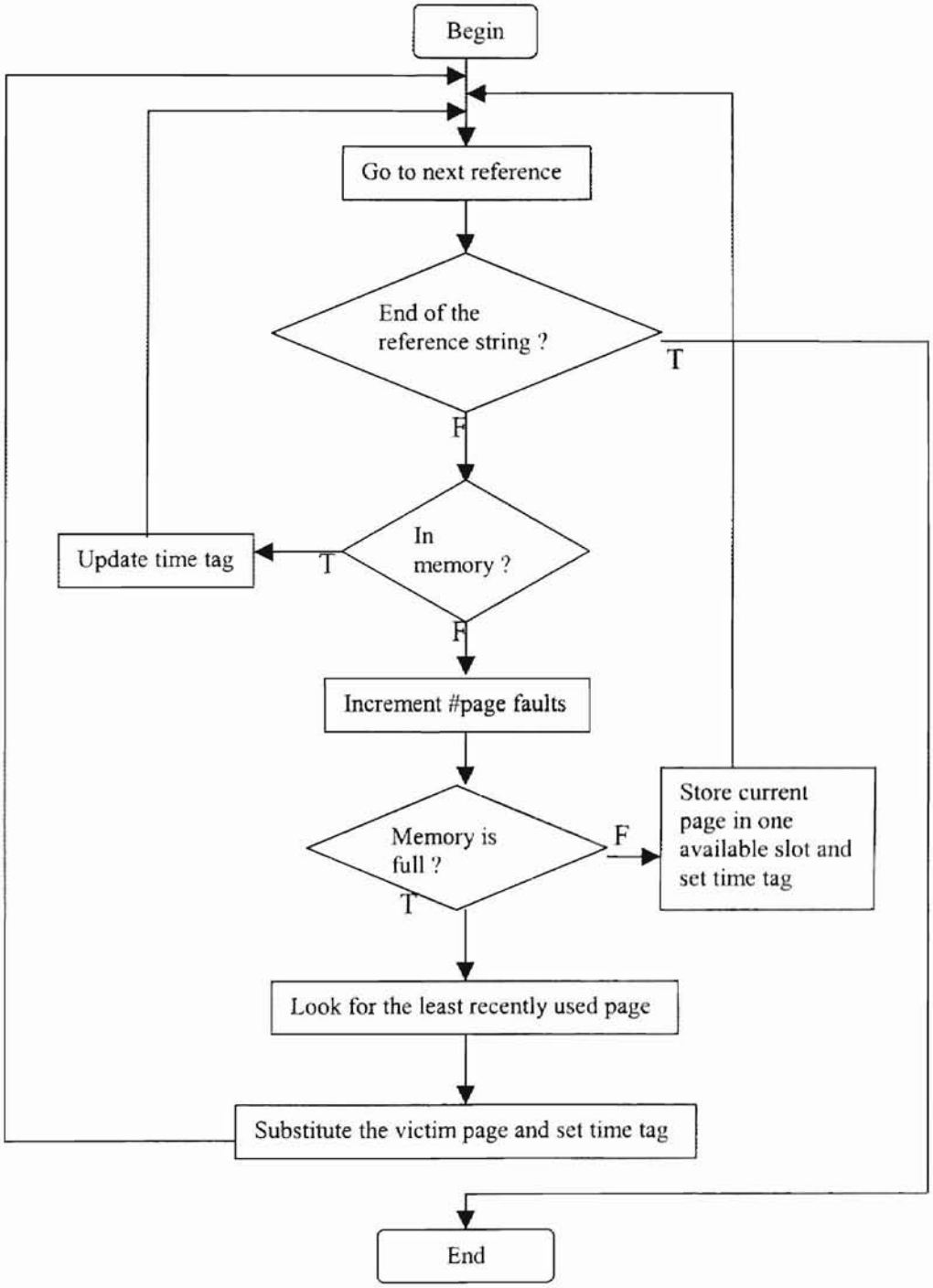
Since x_i are distinct for $1 \le i \le k$, no matter what w is, references to $x_1, x_2, ...,$ and x_k all cause page faults. So after the first loop iteration, the number of page faults is k independent of w. At the second reference to page x_1, w is set to be $k-1$ according to the principle of the proposed algorithm whatever it initially is and it won't change thereafter.

Therefore, initial locality window has no effect on the number of page faults for any reference string in PLM_kn in the proposed algorithm.

APPENDIX F: FLOW CHART OF THE PROPOSED ALGORITHM



APPENDIX G: FLOW CHART OF THE LRU ALGORITHM



VITA

Xiang Hui Liu

Candidate for the Degree of

Master of Science

Thesis: EXPLOITING BOTH SPATIAL AND TEMPORAL LOCALITY IN PAGE REPLACEMENT ALGORITHMS

Major Field: Computer Science

Biographical:

Personal Data: Born in ZunYi, GuiZhou, P.R.China, the daughter of Ronghua Liu and Jinqiu Zeng.

Education: Graduated from ZunYi High School, ZunYi, GuiZhou, P.R.China in July, 1990; received Bachelor of Science degree in Mathematics from Peking University, Beijing, P.R.China in July, 1995; Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December, 2000.

Experience: Employed as a software engineer by the People's Bank of China in NingBo, P.R.China from July, 1995 to December, 1997; employed by Oklahoma State University, Department of Mathematics as a teaching assistant from January, 1998 to August, 1999; employed as a summer intern during summer, 1999; employed as a research assistant by Department of Microbiology and Molecular Genetics and School of Mechanical and Aerospace Engineering, Oklahoma State University from September, 1999 to August, 2000.