1994

A Self-Adaptive Database Buffer Replacement Scheme.

Yueguo Tong
Louisiana State University and Agricultural & Mechanical College

Order Number 9508607

A self-adaptive database buffer replacement scheme

Tong, Yueguo, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A SELF-ADAPTIVE DATABASE BUFFER REPLACEMENT SCHEME

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Interdepartmental Program in Business Administration

by
Yueguo Tong
B.S., Fudan University, 1985
M.S., Fudan University, 1989
August 1994

ACKNOWLEDGEMENTS

First of all, I must thank Dr. Ye-Sho Chen, my major professor, for his invaluable guidance, suggestions, and support throughout this research. Without his consistent help over the years, it would have been impossible for me to reach this point.

I should also thank members of my doctoral committee, Dr. Peter Kelle, Dr. Ishwar Murthy, Dr. Sumit Sarkar and Dr. Kwei Tang for their guidance and helpful suggestions. Especially, Dr. Sarkar's helpful suggestions made this research more worthwhile.

My thanks to my friends at LSU who made my life here enjoyable; to Pete who helped proofreading the manuscript; and to many who offered me various assistance over the years.

My special thanks to my family: to my mother who travelled thousands of miles to help me during the difficult times; to my father who had been proud of me till his last minute in the hospital; to my little daughter whose first several days in this world gave me a peace of mind; and to my beloved wife, from whom I have been getting encouragement and support ever since my years in college. Without her, I would not have reached thus far.

TABLE OF CONTENTS

ABSTRACT

The overall performance of a database system is very sensitive to the buffer replacement algorithm used. However, the performance evaluation of database buffer replacement algorithms commonly assumes that database accesses are independent and the probability for each individual database record to be accessed is fixed. Due to these rigid assumptions, the results of performance evaluation are not always reliable. In this dissertation, we apply Simon's model of information accessing to model database accessing frequencies. This approach relaxes the independent assumption, and since it also allows certain dynamic behavior in accessing frequencies; thus, it is more robust and preferable over the traditional "artificial data" approach. Furthermore, taking advantage of the conceptual similarity between the self-organizing linear search heuristics and the traditional buffer replacement algorithms, we propose a self-adaptive buffer replacement scheme that outperforms conventional database buffer replacement algorithms. The findings of our study can be further applied to many other computer applications, e.g. the more complex problem of archival storage design in larger database systems.

CHAPTER 1 INTRODUCTION

Under current economic conditions, doing more with less seems to be the motto of many organizations. As a critical component of any organization, information systems (IS) need to be designed and implemented to require a minimal amount of time and effort and gain maximum benefits. This chapter gives an overview of the principle of database buffer replacement and self-organizing heuristics for improving the performance of IS. The need for modeling the frequency of information accessing in performance analysis is discussed, followed by an explanation of problems to be addressed in this dissertation. Finally, the dissertation purpose and organization are outlined.

1.1 Database Buffer Replacement

A prevailing problem in database processing is the efficiency in storage, manipulation and retrieval of data. Because of the large value the systems have to deal with, external storage devices (e.g. disks and tapes) are used to organize the mass data. However, under most commercially available operating systems, data can only be manipulated in computer's main memory (Effelsberg and Haerder 1984). Thus, in order to access any database records, the part of the disk storage (page) corresponding to the records requested

must be present in main memory. For this purpose, a database management system (DBMS) maintains an area in the main memory known as the database buffer.

To facilitate the exchange of data between external storage devices and the main memory, a database is divided into pages of equal size with the database buffer correspondingly consisting of page frames of the same size (Effelsberg and Haerder 1984). Any data accessing request is processed by first searching the buffer, assuming that the needed data is already there. Suppose the data is not found in the buffer, a buffer fault occurs; and subsequently the corresponding data page containing the requested data is brought into the buffer from the external storage devices. This procedure is illustrated in Figure 1, where X is the requested page in the secondary storage and Y is the one in the buffer memory which will be replaced by using some selected page replacement algorithm.

Generally, a DBMS manages the data buffer by deciding which page to bring into buffer, where in the buffer to put the page, and, which data page in the buffer is to be replaced by the data page just brought in. All these operations are necessary because the buffer space is limited. As a result, usually only a small portion of a database can be retained in the buffer area at any given time. Database buffer replacement algorithms are

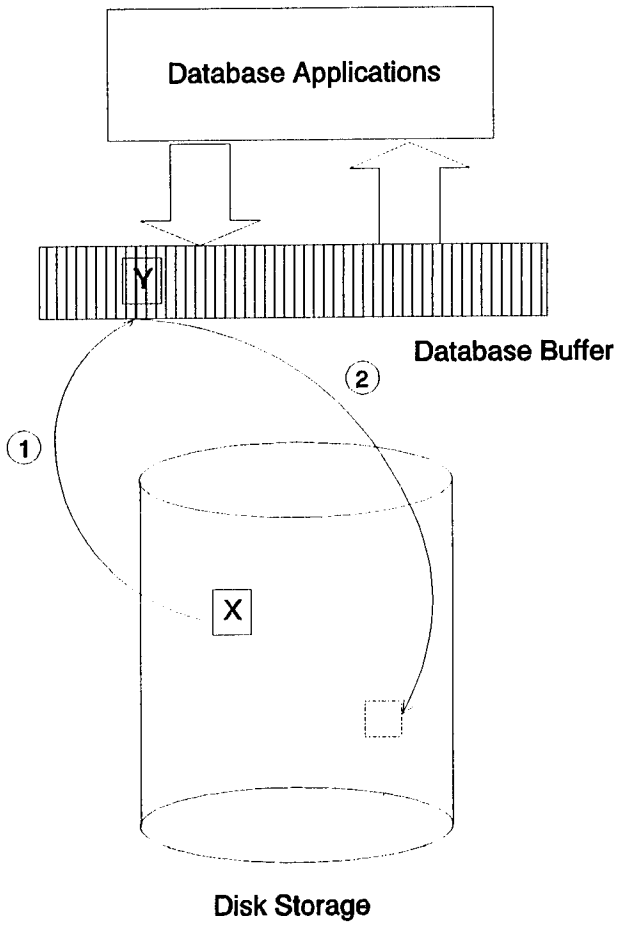

Figure 1: Illustration of Database Buffer Replacement

essentially the rules specifying which data page will be replaced when a fault occurs and the buffer is full. The efficiency of buffer replacement algorithms plays a critical role in overall database performance, and it is the focus of our study in this dissertation.

The procedure of database buffer replacement described above indicates that servicing data requests requires a DBMS to perform input/output (I/O) operations routinely. It is well known that I/O activity is among the slowest operations in a computer system (Palvia 1991). Physical access to a database page on a disk is much more expensive than accessing the same data in a buffer. It is reported (Effelsburg and Haerder 1984) that such an operation not only costs 25 to 50ms of disk access time, but also involves 2000 to 5000 CPU instructions in most operating systems environment. Therefore, the general objective of database buffer replacement is to minimize the physical I/O. Consequently, database buffer replacement algorithms should avoid replacing those database pages that application programs will most likely access in the near future. It is thus obvious that a database buffer replacement algorithm has significant impact on the performance of database systems.

Similar principles have been used frequently in storage management in operating systems (Silberschatz et al. 1991;

Lau 1982) as well as cache memory in computer systems (Hossain et al. 1991). For example, in a virtual memory (VM) environment, the prime objective is to keep the most frequently used pages in the main memory. This will also avoid I/O processing (page fault) time for these data. In case of a page fault, most commercial operating systems (for example, the BSD UNIX on VAX) use the so-called Least Recently Used (LRU) replacement algorithms (Stonebraker 1981). The details of the LRU algorithm and various other replacement algorithms will be described in the next chapter.

1.2 Self-Organizing Heuristics

The principle used in designing buffer replacement algorithms has a very close relationship with the so-called self-organizing heuristics – or permutation algorithms discussed in this section. While data in a database environment are typically kept in a two-level storage system and exchanged between the main memory and the external storage for processing, the data records in this section all reside in main memory.

Given an information retrieval situation where accessing probability of each data element is unknown in advance, it is often effective to dynamically reorganize the data structure when the access process is being performed. Self-organizing heuristics are rules specifying how to

reorganize data structure after each access. The objective is to improve the efficiency by moving data elements that are likely to be accessed in the immediate future closer to the beginning of the structure where searches begin.

Generally, the data described here can be organized as a linear list (such as an array, a singly linked list, a doubly linked list), or some nonlinear data structures (such as binary tree, multiway search tree, and splay tree) (Sleator and Tarajan 1985a). There have been many different self-organizing heuristics reported in the literature, designed for the various type of data structure used. While the majority of research use linear search self-organizing heuristics, the so-called self-adjusting algorithms for multiway search trees (Hui and Martel 1993, Martel 1991), the rotation operation in binary search tree (Cheetham et al. 1993), and the node alignment strategies (Naor et al. 1991) (used to implement the event-list in discrete event simulation programs) are essentially extensions of self-organizing linear search heuristics. In this dissertation, we focus on linear data structure cases since the linear search self-organizing heuristics are conceptually most similar to the buffer replacement algorithms we study. The study of self-organizing linear search heuristics in turn provides us some insights to the buffer replacement algorithms.

Note that in a general linear search procedure, records of data are initially unordered and sequentially organized as either a linked list or an array structure. Each search always starts from the beginning of the list (array) and progresses linearly by comparing a requested key with a key value associated with each record until the desired record is found or the end of the list (array) is reached. In practice, however, it is seldom the case that all records in a list are equally likely to be accessed. It can be observed that some records are accessed more frequently than the others. In other words, the accessing frequencies are skewed.

The idea of self-organizing heuristics is to take advantage of this skewed access frequencies and keep those more frequently accessed records as close to the start of the data structure as possible so the average search time can be minimized. This enhancement to a simple linear search (sometimes also called permutation algorithms) is implemented by moving the accessed records forward a pre-specified distance according to a heuristic rule selected. In addition, each heuristic rule specifies the number of accesses before the permutation is performed. Various heuristic rules differ in the distance the accessed record moves forward and the frequency of permutation. Detailed

descriptions of some heuristics will be given in the next chapter.

One direct application of self-organizing linear search in computer science is a list of identifiers maintained by a compiler or interpreter (Bitner 1979). The list cannot be initially ordered since access frequencies are not known in advance. However, since most programs tend to access some identifiers more frequently than the others, a compiler can improve efficiency by moving the more frequently accessed identifiers closer to the front of the list, using some type of self-organizing heuristics. The improvement of the efficiency was observed from the LISP system at the University of California at Irvine (Hester and Hirschberg 1985).

Bently and McGeoch (1985) described a case in which the application of self-organizing linear search significantly improved the performance of a VLSI circuit simulator. On typical runs of the circuit simulator, the first phase which read the description of the circuit took five minutes before the actual simulation began. The most time consuming part of phase one was the sequential searches in the simulator's symbol table. By using Move-To-Front heuristics (defined in the next chapter), a typical run of the first phase would only take thirty seconds instead of five minutes. The cost

was merely adding several lines of code to incorporate the Move-to-Front heuristics.

In both applications described above, the combination of self-organizing heuristics and linear search has satisfied the needs in each case. The self-organizing linear search has been shown to be effective and the principle of this method can be applied to many other applications, including the database buffer replacement algorithms as will be discussed later in this dissertation.

1.3 The Need for Modeling the Access Frequencies

Before we start discussing any specific buffer replacement algorithms, the reference (accessing) behavior of database transactions has to be considered first, because the key to designing an optimal replacement algorithm is to understand how database transactions access data in databases (Kearns and DeFazio 1989). Similar requirement is also applicable in discussing the self-organizing heuristics.

To simplify the terms used in this dissertation, we define a record to be a general data unit in a database environment. It may be a tuple in a table of a general relational database, or a page in a database buffer or secondary storage. We will use these terms interchangeably in this dissertation.

Considering the literature of database accesses (Kearns and DeFazio 1989), one important observation is that database accessing patterns can exhibit both localized and sequential reference behavior. In other words, the accessing pattern may not be stable over the entire accessing sequence, but may display some localized phenomenon. Also, it is common for records to be accessed in physical sequential order. Generally speaking, these characteristics differ substantially from reference behaviors normally found in program memory references; therefore, effective buffer replacement strategies cannot be based on simplistic assumptions of reference properties such as locality alone or independent accesses as used in program memory references.

However, many rigid assumptions were used in the literature of analyzing and comparing the performance of buffer replacement algorithms and self-organizing heuristics. Since the results of evaluations heavily depend on the realism of the record access frequencies (Fenwick 1991), it is crucial to study the suitability of the existing modeling methods and identify a sound access frequencies representation scheme.

1.4 Problem Statement

As we know, frequent accesses to the secondary storage cause long latency period in database transactions. This

degrading of performance in databases offers us compelling motivation for the study and development of effective buffer replacement strategies which lessen the system's I/O load to increase the throughput of the overall system.

The conventional database access models lack the capability to extract and represent the dynamic aspects of the changing world. We therefore should first identify a more realistic modeling method. Our research attempts to answer the following questions: (1) How do we model the database access frequencies? (2) What impact does an application's access frequencies have on the performance of self-organizing heuristics as well as buffer replacement algorithms? (3) How to define a database buffer replacement algorithm that is best suited to the applications concerned, given that applications have various access patterns due to their own changing environments?

With regard to the first question, several approaches have been proposed. These include the 80/20 rule, empirical usage distributions, and usage process approach. The usage distribution approach, one that is most frequently used in the literature, applies some empirical distributions (e.g. Zipfian distribution, and Lotka's distribution). However, it also has certain unavoidable limitations. Some of these are (Chen 1988): (1) estimating the parameters of a distribution is difficult and sometimes impossible; (2) the

necessary statistical theory for making rigorous test is not available; and (3) assumptions such as independent access probability are mostly unrealistic in many application environments.

To avoid these limitations, we propose the usage process approach originally proposed by Herbert Simon (1955). With extensive study and computational experiments, we find this modeling method to be a constructive approach, and it provides us a realistic and more accurate performance evaluation of self-organizing heuristics and buffer replacement algorithms. The results from our initial studies in turn lead us in identifying a better database buffer replacement scheme.

1.5 Dissertation Purpose and Organization

The purpose of this dissertation is threefold. First, we apply Simon's model of information accessing that better describes the access frequencies in studying database buffer replacement algorithms and self-organizing heuristics. Second, we study some of the self-organizing linear search heuristics reported in the literature under Simon's model. Tighter bounds of relative performance are obtained and the results are logically extended to study the buffer replacement algorithms. As a result, we propose a new class of buffer replacement algorithms that is simple to implement yet more general and flexible than traditional algorithms.

The performance of our newly proposed algorithm is investigated and compared with other existing algorithms, and certain performance advantages are identified in our dissertation. Finally, we further extend our study of buffer replacement and self-organizing heuristics and propose a self-adaptive buffer replacement scheme which has obvious performance advantages over the traditional database replacement algorithms. The implementation and applications of this replacement scheme is also outlined.

This dissertation is organized as below. Chapter 2 gives a comprehensive literature review of current studies in database buffer replacement and self-organizing heuristics, as well as the modeling methods for information accessing in these fields. Certain problems and limitations of existing evaluation methods are identified. Chapter 3 first discusses some necessary adjustments to Simon's autoregressive model and then apply the adjusted model to evaluate the performance of self-organizing heuristics. Chapter 4 presents the computational examination of the buffer replacement algorithms in Simon's model of information accessing, including the newly proposed replacement algorithm. Chapter 5 discusses the development and implementation of a self-adaptive buffer replacement scheme, which is the direct result of studies from Chapter

3 and Chapter 4. Finally, Chapter 6 is the conclusion and the discussion of future research.

CHAPTER 2 LITERATURE REVIEW

In this chapter, we give an extensive review of the literature of database buffer replacement and self-organizing heuristics. While previous research works in these two areas are mostly independent, we identify and summarize some of the common problems that remain unsolved thus far in these two related fields.

2.1 Buffer Replacement Algorithms

Most database systems use a main memory area as a buffer to reduce accesses to disks. This buffer area is subdivided into frames, and each frame can contain a page of data in secondary storage (Kearns and DeFazio 1989). An application requesting a page will cause a fault if the page is not in the buffer. Consequently, this requested page is read into an available buffer frame if buffer space is available. However, if no available frames exist in buffer, a buffer replacement algorithm must be applied to decide which of the buffer pages has to be replaced to make frame space available.

There has been abundant of literature in the area of buffer replacement in databases (Berg and Towsley 1993; O'Neil et al. 1993; Nicola et al. 1992; Hossain et al. 1991; Dan and Towsley 1990; Ch'ng et al. 1989 and 1987; Casas and

Sevcik 1989; Kearns and DeFazio 1989; Sacco and Schkolnick 1986; Effelsberg and Haerder 1984; Stonebraker 1981; Smith 1978; Lang and Fernandez 1977). The buffer replacement algorithms studied in this dissertation belong to a group called demand paging algorithms. These replacement algorithms fetch only the requested page when there is a page fault. In other words, a page is read into buffer only if there is a request for the data on this page. Because of their importance in practical applications, we will discuss and investigate them in this dissertation.

Obviously, the simplest and the most straightforward replacement algorithm is the so called random replacement (RR) algorithm. As the name explains, this algorithm works by randomly selecting one page in buffer and removing it into secondary storage. As we can imagine, this algorithm is not efficient in almost all cases since it does not use any possible information of access sequence. The result of work by Ch'ng et al. (1989) confirmed our conjecture.

Efficient algorithms should replace the buffer page having the lowest probability of reference. They usually rely on the history of references in order to estimate future reference behavior. One general assumption in using these algorithms is that recent reference behavior is a good indicator for the near future. Hence the age and the

reference frequencies of a buffer page can be applied as suitable criteria to predict future reference behavior.

Various replacement algorithms can be classified by whether or not the following considerations are reflected in each algorithm's page selection decision (Effelsberg and Haerder 1984): (1) age since the page's first reference, or since the last reference; and (2) all references, or only the most recent reference to the page.

Simple intuitive consideration lead to the conclusion that the exclusive use of only one of these criteria cannot guarantee an optimal replacement decision, especially when the reference pattern of database varies significantly under different applications (Effelsberg and Haerder 1984). Therefore, various buffer replacement algorithms were reported in the past (O'Neil et al. 1993; Nicola et al. 1992; Hossain et al. 1991; Ch'ng et al. 1989 and 1987; Effelsberg and Haerder 1984). At the same time, a lot of work focus on studying the performance and applicability of these algorithms.

2.1.1 Definitions

2.1.1.1 First-In, First-Out (FIFO)

The First-In, First-Out algorithm (Belady 1966) replaces the oldest buffer page (the one which was earliest to enter buffer) without considering its reference

frequency. The age of a page's first reference is the only decision criterion.

As an illustrative example of the FIFO algorithm, assume a buffer which can hold 3 pages is initially empty. If an access sequence is to page A, B, B, C, D, B, A, C respectively, the change of data pages in buffer and the occurrence of page fault will be exactly as shown below:

No.	Page	Buffer after Access	Replacement (#)	Page
1	A	[A]	No (0)	
2	B	[A B]	No (0)	
3	B	[A B]	No (0)	
4	C	[A B C]	No (0)	
5	D	[B C D]	Yes (1)	A
6	B	[B C D]	No (1)	
7	A	[C D A]	Yes (2)	B
8	C	[C D A]	No (2)	

2.1.1.2 Least Frequently Used (LFU)

The Least Frequently Used algorithm (Coffman and Denning 1973) uses only the second decision criterion described previously, without considering the age of accesses. It replaces the buffer page with the lowest reference frequency. In case there are frequency ties among several pages, the one which enters the buffer most recently will be selected to be replaced. Since LFU has no means to discriminate a page's recent versus past reference

frequency, it is unable to cope with evolving access patterns. Given the same access sequence as in the last section, the change of pages in buffer and the occurrence of page fault will be exactly as below:

No.	Page	Buffer after Access	Replacement (#)	Page
1	A	[A^1]	No (0)	
2	B	[A^1 B^1]	No (0)	
3	B	[B^2 A^1]	No (0)	
4	C	[B^2 A^1 C^1]	No (0)	
5	D	[B^2 A^1 D^1]	Yes (1)	C
6	B	[B^3 A^1 D^1]	No (1)	
7	A	[B^3 A^2 D^1]	No (1)	
8	C	[B^3 A^2 C^2]	Yes (2)	D

2.1.1.3 Least Recently Used (LRU)

The Least Recently Used algorithm (Mattson et al. 1970) considers age of references as well as the frequency of references. It replaces the buffer page that was least recently used, in other words, the page whose most recent access was earliest. This algorithm can be illustrated using the similar example below:

No.	Page	Buffer after Access	Replacement (#)	Page
1	A	[A]	No (0)	
2	B	[B A]	No (0)	
3	B	[B A]	No (0)	
4	C	[C B A]	No (0)	

No.	Page	Buffer after Access	Replacement (#)	Page
5	D	[D C B]	Yes (1)	A
6	B	[B D C]	No (1)	
7	A	[A B D]	Yes (2)	C
8	C	[C A B]	Yes (3)	D

LRU is the most popular replacement algorithm. Moreover, the LRU strategy is simple and can be very efficiently implemented. This is especially important because the buffer management in databases is one of the most heavily used system components. Although it is commonly believed that LRU is a generally good algorithm for buffer memory management, Stonebraker (1981) indicates that LRU appears to perform only marginally in a database environment, in other words, the use of LRU in database buffer does not guarantee the optimal performance.

2.1.1.4 Optimal Replacement (OPT)

The optimal replacement algorithm (Belady 1966) will result in a minimum I/O among all algorithms under the same database access. The algorithm simply replaces the page that will not be used for the longest period of time, or, in other words, the longest future reference distance. However, this algorithm is based on the knowledge of future reference string and thus is not practical (Effelsberg and Haerder 1984). On the other side, the optimal performance is of theoretical interest since it can be used to derive a

bound of performance for a given reference string. Given the same example above, when fault happens at reference number 5, page C is replaced since page C will not be used until access number 8, while B and A will be used at access number 6 and 7 respectively, as displayed below.

No.	Page	Buffer after Access	Replacement (#)	Page
1	A	[A]	No (0)	
2	B	[B A]	No (0)	
3	B	[B A]	No (0)	
4	C	[C B A]	No (0)	
5	D	[D B A]	Yes (1)	C
6	B	[D B A]	No (1)	
7	A	[D B A]	No (1)	
8	C	[C ? ?]	Yes (2)	*

Note that the * above indicates the page that was replaced depends on the knowledge of future accesses. The optimal replacement algorithm guarantees the lowest possible miss ratio for a fixed buffer size.

2.1.2 Performance Measurements

The criteria in evaluating the performance of buffer replacement algorithms are based on the number of I/O operations actually incurred in the database systems as a result. This is chosen because it is the most significant contributor to the overall performance of database systems.

The goal of each replacement algorithm is a minimization of the buffer fault rate for a given buffer size. In contrast to OS replacement algorithms, which are supported directly by special hardware features, those for DBMS buffer management have to be fully accomplished in software. This situation, however, offers more freedom with respect to the choice and evaluation of selection criteria.

The performance of database buffer management is measured by a metrics called Miss Ratio. It is defined as the steady-state probability that the record currently being accessed is not in main storage and is estimated through calculating the ratio of the number of accesses that causes fault to that of the total number of accesses. For example: a sequence of 8 accesses to the buffer incurs 2 faults, thus the Miss Ratio is 2/8=0.25. Another performance measurement which has a close relationship with Miss Ratio is so called the Hit Ratio. It can be calculated directly from Miss Ratio by the formula:

Hit_Ratio = 1 - Miss_Ratio

Much of the work in this area is concerned with predicting the value of these measurements, designing the practical algorithm for maximizing (minimizing) these metrics, or choosing the size of buffer so as to maximize the hit ratio subject to some cost constraints (Berg and Towsley 1993). The result of increased buffer hit

probability reduces the average disk I/O per transaction, and thus improves the transaction time as well as the overall system throughput.

2.1.3 Current Findings

There has been a significant amount of literature in the buffer replacement study (Berg and Towsley 1993; O'Neil et al. 1993; Nicola et al. 1992; Hossain et al. 1991; Dan and Towsley 1990; Ch'ng et al. 1989 and 1987; Casas and Sevcik 1989; Kearns and DeFazio 1989; Sacco and Schkolnick 1986; Effelsberg and Haerder 1984; Stonebraker 1981; Smith 1978; Lang et al. 1977). The study in this field has focused on two directions: (1) the evaluation of existing algorithms under various access sequences; and (2) design and implement new efficient algorithms.

Effelsberg and Haeder (1984) classified various replacement algorithms. According to their discussion, FIFO is only appropriate for sequential access behavior and the pure LFU mechanism should not be implemented in a database environment. However, they believed that by using some additional mechanism, the LFU concept can be made more appropriate by using some additional mechanism, while losing its original characteristics. The LRU algorithm which considers age as well as reference is generally preferred in DBMS according to their study^T ^\%^R+^\other replacement algorithms were also proposed and studied in their paper.

A comparison of different replacement algorithms was performed based on the two reference strings of different transaction load of a CODASYL DBMS from a school database application. As they indicated, comparable algorithms deserve further investigation when applied to a specific DBMS, in other words, the "best" algorithm they identified may not be the best universally.

Ch'ng et al. (1987 and 1989) studied various existing buffer replacement algorithms under skew access patterns, generated from a simplified Zipfian distribution. Their results show that FIFO and Random Replacement (RR) are the least effective among four algorithms studied with increasing skewness. They indicated that TRANS algorithm, which can be considered as a variation of the LRU algorithm (detail discussed later) is better than the more popular LRU. As a result of their study, they proposed a hybrid algorithm called Double Area which they claimed improving the performance significantly.

Dan and Towsley (1990) developed approximate analytical models for predicting the buffer miss ratio under the LRU and FIFO buffer replacement under the independence reference model. To demonstrate the usefulness of their models, they considered two applications and compared the LRU and FIFO algorithms with the OPT algorithm. Their result indicated that the performance of LRU is close to that of OPT, and

always perform better than FIFO algorithm. Properties of the miss ratio functional form of the LRU and FIFO algorithms were also analyzed and reported by Berg and Towsley (1993) under some assumptions of the reference model.

Most recently, O'Neil et al. (1993) proposed a new algorithm called LRU-K for database buffer replacement. Their basic idea is to keep track of the times of the last K references to popular database pages, using this information to statistically estimate the inter-arrival ti^\^U^R f references on a page by page basis. They demonstrated that the LRU-K algorithm surpasses conventional algorithms in discriminating between frequently and infrequently referenced pages, and is believed to be self-tuning and does not rely on external hints about workload characteristics. In addition to LRU-K, fuzzy replacement algorithm (Hossain et al. 1991) was also proposed recently in the literature.

2.2 Self-Organizing Heuristics

Self-organizing heuristics are essentially permutation algorithms that maintain self-organizing lists by rearranging data elements. To describe various self-organizing heuristics, we follow the definition of Hester and Hirschberg (1985), and define an accessed record as the record currently being searched for. Some commonly used

self-organizing linear search heuristics that will be studied in our research are described in this section.

2.2.1 Definitions

2.2.1.1 Move-To-Front (MTF)

Under Move-To-Front heuristic rule, when an accessed record is found, it is moved to the front of the list. All the records that are originally in front of the accessed one are moved back one position to make room. Thus, it is easy to know that after the permutation the order of all the other records in the list is unchanged relative to each other. Also, if the first record is the accessed record, the list will stay the same. In fact, these two observations will be true for all the heuristic rules under our discussion.

As an illustrative example of MTF heuristics, assume a list has the form [A B C D E F] initially, where the left most record A is the beginning of the list. If an access sequence is A, C, F, A, B, the change of relative positions of each data record will be exactly as shown below:

Access No.	Accessed Record	List after Accessing
1	A	[A B C D E F]
2	C	[C A B D E F]
3	F	[F C A B D E]
4	A	[A F C B D E]
5	B	[B A F C D E]

2.2.1.2 Transpose (TR)

In this heuristics, after each access, the accessed record, if not at the front of a list, is moved up one position closer to the front of the list by swapping the positions with the record directly in front of it.

If we assume the same initial list structure and the access sequence as in last section, the change of positions of each data record will be:

Access No.	Accessed Record	List after Accessing
1	A	[A B C D E F]
2	C	[A C B D E F]
3	F	[A C B D F E]
4	A	[A F C B D E]
5	B	[A F B C D E]

2.2.1.3 Move-Ahead-K (A_k)

Move-Ahead-K was initially proposed by Rivest in 1976. Since then further research work has been done by Gonnet et al. (1981), Ch'ng et al. (1989), etc. This heuristics is more general and can be looked as a compromise between the relative extremes of Move-To-Front and Transpose. Under this heuristics, after each access, the permutation rule simply moves the accessed record forward K positions or to the front of the list if its current location is less than K from the front of the list (Rivest 1976). Let N be the total number of records in a list, then MTF is equivalent to

A_N and TR is exactly the same as A_1. Given the same list assumptions above, and assume K=2, the change of the list will be:

Access No.	Accessed Record	List after Accessing
1	A	[A B C D E F]
2	C	[C A B D E F]
3	F	[C A B F D E]
4	A	[A C B F D E]
5	B	[B A C F D E]

Notice that the computer implementations of all the three heuristics described so far require no storage other than that for records of a list. These heuristics are therefore called memoryless algorithms.

2.2.1.4 Count (C)

With this heuristics, a frequency counter is used for each record in a list to keep track of its access frequency. After each access, the frequency count of the accessed record is incremented by one (an integer that is initially set to zero) and this record is moved ahead of all records with lower counts. In other words, records in the list are always kept in descending order of their cumulative number of accesses.

If we assume all the data records in the previously assumed list have initial access frequency of 1, then the change of record order in the list as well as their

frequency counters (represented as the superscript of each element) will be:

Access No.	Accessed Record	List after Accessing
1	A	$[A^2\ B^1\ C^1\ D^1\ E^1\ F^1]$
2	C	$[A^2\ C^2\ B^1\ D^1\ E^1\ F^1]$
3	F	$[A^2\ C^2\ F^2\ B^1\ D^1\ E^1]$
4	A	$[A^3\ F^2\ C^2\ B^1\ D^1\ E^1]$
5	B	$[A^3\ F^2\ C^2\ B^2\ D^1\ E^1]$

One thing important here is that the implementation of this heuristics' requires substantial additional space for frequency counters, which, as Knuth (1973) remarks, could perhaps be better used by employing non-sequential search techniques. Because of the extra space requirement, Count heuristics has been considered in a different class from previously discussed types and received relatively less attention in the literature (Bently and McGeoch 1985).

2.2.2 Performance Measurements

To define what is meant by a "good" heuristics, two important performance measurements discussed in the literature are asymptotic cost and the rate of convergence (Hester and Hirschberg 1985). A relative measurement is sometimes used to compare the performance among different heuristics. Notice that in these measurements, the cost of a permutation algorithm depends on a given search sequence and the initial list ordering. It is generally defined as

the average cost per access in terms of the number of probes (comparisons) required in finding accessed records, plus the amount of work required to rearrange the records afterward.

2.2.2.1 Asymptotic Cost

Asymptotic cost of self-organizing heuristics is the average cost over all initial list configurations and search sequences without considering the cost of rearranging the data structure. This cost can be measured in terms of the average number of probes required to find the accessed records. For example, if a search sequence of five accesses requires respectively 1, 3, 6, 1, 4 probes to locate each of its data records, the average cost in this case will be: (1+3+6+1+4)/5 = 15/5 = 3.

2.2.2.2 Rate of Convergence

It is expected that the result of applying self-organizing heuristics will cause records that are more frequently accessed to be moved closer to the front of the list. It is unreasonable, however, to expect any list to ever converge to and remain at a perfect ordering according to the values of access probabilities. In fact, the use of heuristics is expected to reach a steady state where many further permutations are not expected to change the expected search cost significantly. Rate of Convergence is used to measure how quickly a permutation rule approaches its steady state.

Bitner (1979) proposed a measure of convergence, called Overwork. It is defined as the area between two curves on a graph, illustrated in Figure 2 (Hester and Hirschberg 1985). It shows the average cost as a function of the number of records accessed. Obviously, as accesses are made and records are self-organized, the curve should indicate that the average number of probes per access (average cost) will decrease as shown in Figure 2, if the heuristics is effective. After a sufficient large number of accesses, the average number of probes reaches an asymptotic cost (the horizontal line in Figure 2). Thus, a smaller area of overwork implies that a heuristics has a faster convergence to its asymptotic cost.

2.2.2.3 Relative Measurements

Relative measurements are often desirable when it is difficult to find an absolute measure of self-organizing heuristics, or when no direct measure can be obtained of a given heuristics. The most commonly used base of comparison is the so called cost of optimal static ordering. This cost is measured when records are initially ordered by their static probabilities of access (assume they are available) and left in that order throughout the entire access sequence. For example, if the cost of optimal static ordering is 2.5 and the asymptotic cost of MTF under the

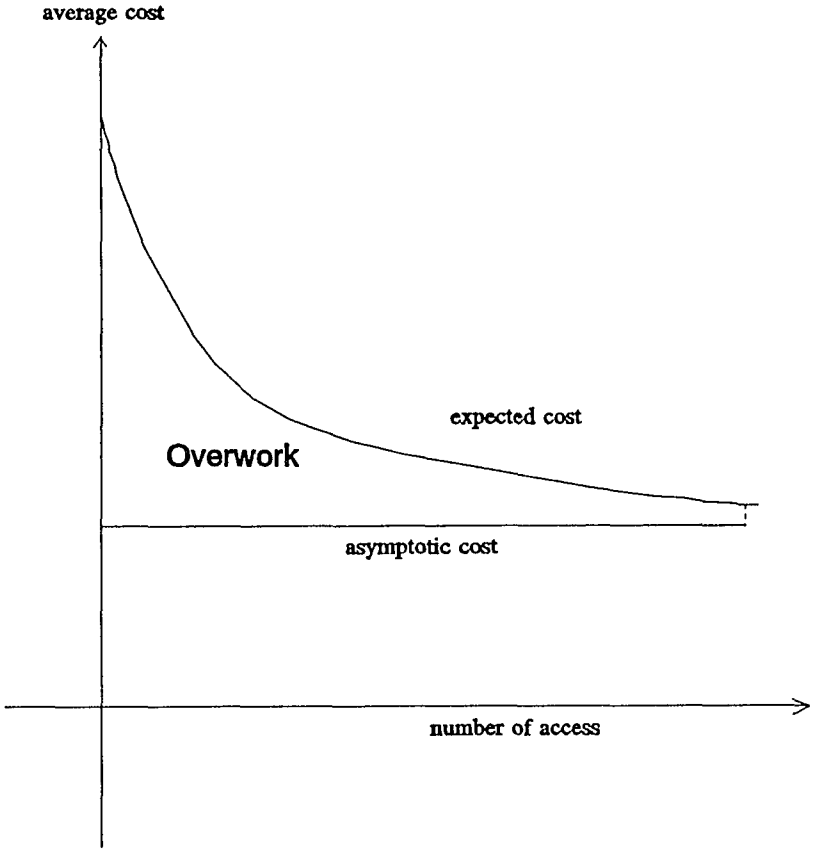

Figure 2: Average Cost and Asymptotic Cost

same access sequence is 3.5, the relative measurement under the above definition will then be 3.5/2.5=1.4.

Notice that the optimal static ordering is not a heuristic algorithm by our definition. The reason is that it uses knowledge about record accessing probabilities that are assumed to be unavailable before searches begin.

2.2.3 Analytical Findings

There have been many analytical studies of self-organizing heuristics in the literature. The following analytical results are all based on the assumptions that records in a list have a static access distribution of $P=(p_1, p_2, \ldots, p_N)$, where p_i, $i= 1,2,\ldots, N$, is the fixed access probability of the *i*-th record in a list and N is the number of records in the list.

2.2.3.1 Move-To-Front (MTF)

This is the heuristics that has been most frequently analyzed in the literature. A general observation is that it tends to converge quickly, but has a large asymptotic cost. Intuitively, this happens since every time when a record with a low access probability is accessed, it is moved to the front of the list. As a result, the costs of future accesses to many other records, some of those with higher access probabilities, will increase.

Given a general search sequence with an access distribution of $P=(p_1, p_2, \ldots, p_N)$, the explicit expression

of the asymptotic cost of MTF heuristics, denoted by $A_{MTF}(P)$, has been shown by many researchers (Bitner 1979; Burville and Kingman 1973; Hendricks 1976; Knuth 1973; McCabe 1965; Rivest 1976), to be

$$1 + 2 \sum_{1 \le i \le j \le N} \frac{p_i p_j}{p_i + p_j}$$

Notice that the cost here includes only the search time, which is assumed to be linearly proportional to the number of probes needed to locate a record. This cost may increase significantly if the time to permutate the records, which depends on the type of data structures used, is also included. Gonnet et al. (1981) derived various expected cost equations for MTF heuristics under different assumptions of skew access distributions. They have commonly assumed Zipfian distribution, Lotka's distribution, exponential distribution, and Wedge distribution, respectively on the values of p_i. With these assumptions, the relative comparisons of the expected cost of MTF to that of the Optimal Static Ordering are also reported in the literature (Hester and Hirschberg 1985).

Bitner (1979) provided an explicit expression of Overwork for MTF, denoted by OV_{MTF}, under the static distribution of $P=(p_1, p_2, \ldots, p_N)$. It has the form of:

$$OV_{MTF} = \frac{1}{2} \sum_{1 \le i < j \le N} \left(\frac{p_i - p_j}{p_i + p_j} \right)^2$$

He further proved that OV_{MTF} is less than N(N-1)/4 for every probability distribution P with N>2. For Zipfian distribution, he gave a complex formula, and for large N the formula can be simplified as $OV_{MTV} \approx 0.057N^2$.

The equations above have been used to show that $A_{MTF}(P)$ is, at most, twice the cost of the optimal static ordering, $A_o(P)$ (Biter 1979; Hendricks 1976; Knuth 1973; McCabe 1965; Rivest 1976). Rivest (1976) conjectured that this bound is not tight; Gonnet et al. (1981) presented a distribution that gave a ratio of $\pi/2$ which is believed to be the tightest yet found. Later on, Chung et al. (1988), using Hilbert's inequalities, proved that the average cost required for MTF heuristics under <u>any</u> probability distribution P, is no more than $\pi/2$ times that of the optimal static ordering and this bound is the best possible.

2.2.3.2 Transpose (TR)

It is reported that slower record movement gives Transpose slower convergence. However, its stability tends to keep its steady state closer to optimal static ordering (Hester and Hirschberg 1985). Intuitively, we know that MTF has a high asymptotic cost due to its potential to make big mistakes by moving a record all the way to the front of a list on the basis of a single access. On the other hand, TR avoids this potential error by being far more conservative in record permutation. However, in cases where there is

some significantly different access pattern in certain subsequence as compared to that in the overall accesses sequence (i.e., the locality phenomenon), TR can be too slow to adapt to the changes. In such cases a more radical algorithm such as MTF may be justified.

Rivest (1976) derived the average search time of Transpose as

$$\frac{\Sigma_{\pi}[(\Pi_{i-1}^{N} p_i^{\delta(i,\pi)})\Sigma_{j-1}^{N} p_j \cdot \pi(j)]}{\Sigma_{\pi} \Pi_{i-1}^{N} p_i^{\delta(i,\pi)}},$$

where π denotes a possible ordering of the records, $\pi(j)$ denotes the jth record in that ordering, and $\delta(i,\pi)$ denotes the quantity $i-\pi(i)$, or the distance that record i will be displaced from its original location in π.

As for the relative measurement, Rivest (1976) also showed that the asymptotic cost of Transpose $A_{TR}(P)$ is less than or equal to that of $A_{MTF}(P)$ for every probability distribution P. He also conjectured that TR is the optimal among all permutation rules. The result is further backed up by Bitner (1979). However, Anderson et al. (1982) presented a counterexample to the original conjecture of Rivest (1976) in the form of a heuristics rule that is better than TR under a specific distribution.

Bitner (1979) could not find a general Overwork formula for Transpose and thus only gave a formula for the special case when $p_1=0$ and $p_i=1/(N-1)$, for $i=2,\ldots,N$. He also

manually calculated the values of Overwork for TR under Zipfian distribution, given various N values. The results appear to be of $O(N^3)$, which is again O(N) greater than his Overwork measure for MTF (approximately $0.057N^2$ for large N).

2.2.3.3 Move-Ahead-K (A_k)

Initially proposed by Rivest (1976), Gonnet et al. (1981) proved that if, j>k, Move-Ahead-j converges faster than Move-Ahead-k, but the asymptotic cost of Move-Ahead-j is higher than that of Move-Ahead-k. However, due to the complexity of this more general type heuristics, no explicit expressions were reported in the literature so far.

2.2.3.4 Count (C)

The Count heuristics uses a frequency counter f_i for requests to the *i*-th record. By the law of large numbers, if $p_i > p_j$, then the frequency f_i may be less than f_j for only a finite number of requests. Thus, the search cost under Count asymptotically approaches that of the optimal static ordering and that the cost difference decreases exponentially with time. Bitner (1976) showed that if the distribution P is unknown beforehand, then Count produces the ordering with the lowest expected cost.

2.2.3.5 Summary

Analytical findings under assumptions of various access distributions have been reported by Gonnet et al. (1981),

Hendricks (1973), Bitner (1976, 1982), Kan and Ross (1980), Tenenbaum and Nemes (1982), Chung et al. (1988). Among all these research works reported, the most frequently used skew distribution is Zipfian distribution. Knuth (1973) showed the cost of MTF under Zipfian distribution is bounded by a value of 2ln2 (about 1.386) times the cost of optimal static ordering A_o, which is tighter than the one given by Gonnet et al. (1981) as well as the one given by Chung et al. (1988), due to his specific assumption on the access distribution.

In summary, for any probability distribution P, the above discussed asymptotic expected cost shows that

$$A_{MTF}(P) \leq \pi/2\ A_o(P),$$

$$A_{TR}(P) \leq A_{MTF}(P), \text{ and}$$

$$A_c(P) = A_o(P)$$

and it is also known that these general relations could be further tightened if more specific assumptions of access distribution are given.

2.2.4 Empirical Results

Thus far, the majority of mathematical analysis were performed on the two fundamental types of heuristics: Move-To-Front and Transpose. Certain theoretical analysis was also given to Move-Ahead-K and some other heuristics. In the mean time, there has been significant amount of computational experiments on self-organizing heuristics,

mostly on the more general types of heuristics, e.g., Move-Ahead-K and the performance comparisons of A_k with MTF, TR, etc.

2.2.4.1 Move-Ahead-K (A_k)

Tenenbaum (1978) performed a simulation study on Move-Ahead-K heuristics. He used files of various sizes as search sequences. Based on the result of simulation, he justified the best value of K (among the range he tested) for the list sizes tested. Notice that Tenenbaum's experiments gave no universal optimal value of K. The reason is that given a constant number of accesses, the optimal value of K is closely related with the number of records in the list, which is clearly indicated in his experimentation.

2.2.4.2 Relative Performance of A_k, MTF, and TR

To study the relative performance of self-organizing heuristics, the expected cost of the optimal static ordering needs to be know. Bentley and McGeoch (1985) compared the simulation results by studying several common self-organizing heuristics using files containing programs or text. They assumed that the words in files had probabilities following Zipfian distribution and thus were able to calculate the expected cost of the optimal static ordering. As a result, they concluded that the distribution

of words is closer to the Zipfian distribution than to a normal distribution.

Bellow (1987) proposed an autoregressive model for record access sequence, known as discrete autoregressive process. After analytically studied the performance of MTF under a simple version of his model, simulation experiments were performed to explore the effects of more complicated models on the relative performance of MTF, TR, and A_k (with selected k values). In his simulation, Bellow also assumed that the long run accessing probabilities follow Zipfian distribution. His results provide a solid support to the belief that MTF should be the preferred heuristics when accessing sequences are generated by words in computer programs. It was also observed in Bellow's experimentation that under high locality conditions, A_K with intermediate K closely approximated to the performance of MTF, even when K was not close to the number of records in the list.

Ch'ng et al. (1989) empirically studied A_K heuristics under skewed access frequencies. They adopted a simpler approximation of Zipfian distribution and used a self-defined parameter called skew factor in describing the accessing sequence. With their nonuniform access frequency assumptions, the performance of MTF ,TR, and the more general A_K (with K varies from 1 to 10) were studied. Their results suggested a hybrid permutation rule, which appears

to have better performance for nonuniform access frequencies.

2.2.4.3 Summary

The general observations from simulation studies are that almost any self-organizing heuristics gives a useful improvement in search performance. And secondly, the performance of all types of heuristics is very sensitive to accessing frequency distributions (Fenwick 1991). Variations of a few percent between heuristics are insignificant compared with the variations between record access frequencies. Therefore, a sound performance evaluation can only be obtained when the modeling method of access frequencies is realistic and applicable to the specific application environment.

2.3 Information Accessing Models

Present modeling methods of information accessing are summarized and compared in this section. With the study of the problems among current approaches, we identify the stochastic process approach as our choice of studying buffer replacement algorithms and self-organizing heuristics.

2.3.1 Empirical Phenomena of Information Accessing

2.3.1.1 80/20 Rule

The simplest way to describe information accessing is to assign some kind of concentration measurement to it. The most often cited concentration measurement is the so called

80/20 rule (Sanders 1987). Heising (1953) states that 80/20 rule is a realistic rule of approximation for information retrieval distribution in commercial applications. The rule implies that 80 percent of transactions deal with the most active 20 percent of the data and the same rule applies recursively to this 20 percent. Assuming the accessing distribution is $P=(p_1, p_2, \ldots, p_N)$, where $p_1 \geq p_2 \geq \ldots \geq p_N$, and N is the number of records in a list, the 80/20 rule can be formalized as:

$$\frac{p_1+p_2+\ldots+p_{0.2N}}{p_1+p_2+\ldots+p_N} = 0.80 \qquad \text{for all N.}$$

Although the 80/20 rule has been quite popular among different disciplines, unfortunately, the measure usually deviates from 80/20 (Chen et al. 1993).

2.3.1.2 Skew Distributions

It is reported that access frequencies often display nonuniform distributions, which has been very popular in commercial applications (Ch'ng et al. 1989). Zipf's law of usage distribution (Zipfian distribution) is probably the most well-known and frequently applied skew distribution in modeling the information accessing (O'Neil et al. 1993; Casas and Sevicik 1989; Chung et al. 1988; Anderson et al. 1982; Hendrick 1976; Bentley and McGeoch 1985; Bitner 1976 and 1979; Rivest 1976; Tenenbaum 1978). In his 1949 book *Human Behavior and the Principle of Least Effort*, Zipf stated that "if one takes the words making up an extended

body of text and ranks them by frequency of occurrence, then the rank r multiplied by its frequency of occurrence, g(r), will be approximately constant." In symbolic form,

$$g(r) = br^{-1}, \qquad r = 1,2,3,\ldots$$

where b is a positive constant whose value depends on the type of text (Zipf 1949).

Under the same assumption of the form of accessing distribution above, Zipfian distribution can be formalized as:

$$p_1=\frac{c}{1},\ p_2=\frac{c}{2},\ \ldots\ ,p_N=\frac{c}{N},\ \text{where}\ \ c=\frac{1}{H_N}\ \ \text{and}\ \ H_N=\sum_{i=1}^{N}\frac{1}{i}.$$

A simplified Zipfian distribution using some approximation was also reported in the literature (Ch'ng et al. 1989). The distribution has been shown through simulation experimentation to be a good approximation of the distribution of common files of words (Bentley and McGeoch 1985) and is also frequently used to describe search sequence in studying self-organizing heuristics and buffer replacement algorithms (O'Neil et al. 1993; Casas and Sevicik 1989; Chung et al. 1988; Anderson et al. 1982; Hendrick 1976; Bentley and McGeoch 1985; Bitner 1976 and 1979; Rivest 1976; Tenenbaum 1978).

Casas and Sevicik (1989) studied the reference behavior of an actual production database workload and provided some evidence that page references follow a Bradford-Zipf-like

distribution. They indicated that their observation can be a useful operational hypothesis for the analysis of database systems, particularly for database performance predication.

Another frequently used skew distribution in studying self-organizing heuristics is Lotka's distribution. In his 1926 paper, Lotka examined patterns of scientific productivity among chemists and physicists. He discovered that if he classified this population according to their publication productivity, then the number of chemists who published n papers, denoted by f(n), was approximately a/n^2, for some positive constant a. This is equivalent to:

$$f(n) = an^{-2}, \quad n = 1,2,3,\ldots.$$

To apply Lotka's distribution, the corresponding information accessing probabilities could be formalized as:

$$p_1=\frac{c}{1^2},\ p_2=\frac{c}{2^2},\ \ldots,\ p_N=\frac{c}{N^2},\ \text{where } c=\frac{1}{H_N^{(2)}} \text{ and } H_N^{(2)}=\sum_{i=1}^{N}\frac{1}{i^2}.$$

Some analytical examinations of the performance under the assumptions of these different skew distributions are reported and summarized in literature (Gonnet et al. 1981).

2.3.1.3 The Need for Better Models

Current studies of the performance of buffer replacement algorithms in the literature either assume that records of databases are accessed uniformly, or each record has a fixed access probability and the accesses are mutually independent (Berg and Towsley 1993; O'Neil et al. 1993;

Nicola et al. 1992; Dan and Towsley 1990). In other words, the string of record accesses is modeled as a sequence of identical independently distributed (i.i.d.) random variables – this is known as the Independent Reference Model (IRM) in the literature.

Casas and Sevcik (1989) provided some empirical evidence for the operational hypothesis that database page references are Bradford-Zipf distributed, and they believe that Bradford-Zipf distribution is more realistic assumption about the distribution of page accesses. Later on, Zipfian distribution of reference frequencies were also applied in studying some newly proposed buffer replacement algorithms (O'Neil et al. 1993). Meantime, the independent access assumption (IRM) continues to be applied but many researchers believe that obtaining results from more interesting reference models where the i-th reference may depend in some way on the previous $j>0$ references (the IRM corresponds to this model with $j=0$) are desirable (Berg and Towsley 1993; O'Neil et al. 1993).

Extensive studies have also been conducted on various self-organizing heuristics (Valiveti and Oommen 1993; Hui and Martel 1993; Cheetham et al. 1993; Ng and Oommen 1992; Fenwick 1991; Courcoubetis and Weber 1990; Kim 1990; Chung et al. 1988; Makinen 1988; Bentley and McGeoch 1985; Oommen and Hansen 1985; Sleator and Tarjan 1985; Anderson et al.

1982; Gonnet et al. 1981; Kan and Ross 1980; Bitner 1976 and 1979; Rivest 1976; P.J. Burville and Kingman 1973; Hendrick 1976; Knuth 1973; McCabe 1965). Previous modeling methods of access frequencies in the self-organizing heuristics study in the literature can be classified into two groups: (1) assuming access frequencies follow some types of well defined distributions, for example, Zipfian distribution, with accesses being independent of each other (Chung et al. 1988; Anderson et al. 1982; Hendrick 1976; Bentley and McGeoch 1985; Bitner 1976 and 1979; Rivest 1976; Tenenbaum 1978); (2) using some selected real data files (Bentley and McGeoch 1985, Fenwick 1991), such as a paragraph of text or some source program files.

Notice that both 80/20 rule and the two empirical skew distributions described above only give a static record access distribution. Also, neither approaches allow the dependent phenomenon among the subsequent accesses. As we already know, the dependent phenomenon is a very common characteristic of a real accessing sequence. Thus, a better modelling approach which incorporates the empirical distributions and allow dependent phenomenon in the accessing sequence will be valuable.

One promising and constructive approach to model information accessing frequencies is the process approach. This approach is more general and provides a generating

mechanism to explain the skewness of empirical distributions. Chen and Leimkuhler (1986), using an index approach, showed that empirical distributions such as Lotka's law and Zipf's law are mathematically equivalent under certain conditions. Thus, the task had been reduced to model one of these empirical laws, for example, Lotka's law. This was accomplished when Chen (1989a) derived Lotka's law from Simon's model using the expected values of the Simon-Yule distribution.

2.3.2 Simon's Model of Information Accessing

2.3.2.1 Simon's Modeling Approach

According to Simon (1968), the central question of building and testing stochastic models of skew distribution is the question of proper treatment of extreme hypotheses; i.e., assertions that a particular specific functional relationship holds between the independent and the dependent variables. Simon suggested that theories of stochastic model should rise inductively from data instead of data being collected solely for fitting pre-existing theories.

Instead of testing hypothesis, Simon recommended changing the question to that of *estimation*. He suggested an approach that consisted of combinations of generalization and refinements. Before one can find laws that fit empirical data, one must have appropriate data that look as though a smooth mathematical function could generate them.

First fascinated in 1936 by the regularities exhibited by Lotka's observations, word frequencies, and city sizes, Simon attempted to find laws that would fit these empirical data. Later on, Simon (1955) proposed a five-step modeling process that not only addressed the formulation of Lotka's distribution, but the establishment of other empirical distributions as well.

Simon's five-step modelling process is:

(1) Begin with empirical data, not hypotheses;

(2) Draw a simple generalization that approximately summarizes striking features of the data;

(3) Find limiting conditions under which deviations from a generalization are small;

(4) Construct simple mechanisms to explain the simple generalizations; and

(5) Propose the explanatory theories that go beyond simple generalizations and create experiments for empirical observations.

Based on his theory of modeling, Simon and Van Wormer (1963) proposed a generating mechanism to simulate the information accessing frequencies which approximated those skew distributions.

2.3.2.2 Version I: Constant Entry of "New" Records

Simon's version I model has the following two assumptions:

Assumption I: The probability that the (k+1)-st record accessed will be a record that has not been accessed in the first k accesses, is a constant α; and

Assumption II: The probability that the (k+1)-st record accessed will be a record that has already been accessed i times ($i \geq 1$) is proportional to $i \cdot f(i,k)$, where $f(i,k)$ is the number of records in a list that have been accessed exactly i times in the first k accesses.

The first assumption differentiates two classes of selections: the data records that have not been accessed before ("new" records) and those that have already been accessed ("used" records). The parameter α, therefore, determines whether a record needs to be moved from the class of "no access" to the class of "accessed once." If the selection is deemed to be a "used" one, we determine its selection through the second assumption. This second assumption essentially places proportionally higher probabilities of access to more frequently accessed records. Thus, a record that has been accessed more frequently is more likely to be accessed again.

2.3.2.3 Version II: Decreasing Entry of "New" Records

Simon and Van Wormer (1963) began by assuming the parameter α in assumption I be a constant, thus independent of the number of accesses, k, that has taken place. They noted that Version I model is only a simple generalization that approximately summarizes the striking features of the observed data. Therefore, this version I model of first approximation was further refined later on into Version II by modifying α to be a decreasing function $\alpha(k)$, i.e., there is a probability function $\alpha(k)$, where $0 \leq \alpha(k) \leq 1$, $\alpha(k+1) \leq \alpha(k)$, $k=1,2,\ldots,$ that the (k+1)-st accessing is to a record accessed for the first time. Notice that the parameter α and functional form of $\alpha(k)$ in version I and II models are determined by the environment in which the selection process takes place; some environments espouse new selections while others may discourage them.

2.3.2.4 Version III: A Model with Autocorrelation

The probabilities of accessing to those already accessed records are assumed to be proportional to their previous number of accesses in Simon's version I and II models. However, the information not used has a tendency of being "forgotten." From analyzing firm sizes which have the same skew distribution as the information accessing, Simon refined his first two versions of models to better reflect the reality (Ijiri and Simon 1977). They further

refined version III model and took into consideration the recency of accesses when assigning probability of access for a "used" record.

In terms of information accessing, the identity of each record is maintained from one time period to the next. The selection of the next record is governed by a stochastic process, which depends on not only how much the record has been used before, but also upon *when* these accesses happened. For simplicity in his computations, Simon assumed that only one record is accessed at each time period defined. The probability that a record will be accessed next is assumed to be proportional to a weighted sum of its access history, where the weight of a usage decreases geometrically, at a rate γ, with the lapse of time since its last access.

Simon formalized their second assumption notions as follows: Let $y_j(k)$ be the indicator of the access of the j-th record during the *k*-th time interval, where $y_j(k)$ is either one or zero with one means the record is accessed and zero means it is not accessed. Then the total number of accesses of the *j*-th record at the end of the k-th time interval is simply $\sum_{\tau=1}^{k} y_j(\tau)$. The probability of accessing the *j*-th record during the (k+1)-st time interval can be formalized as:

$$p[y_j(k+1) = 1] = \frac{1}{W_k} \sum_{\tau=1}^{k} y_j(\tau)\, \gamma^{k-\tau}$$

where W_k is the sum of weighted usage of all records, which is a function of time k and is same for all records; γ is the parameter that determines how rapidly the influence of past access on new selection dies out.

Simon selected a unit time interval in his version III model so that there is exactly one record accessed per any one given time period. He further assumed that there is a constant probability, α, that the selected one is a "new" record. Simon's simulation results closely resemble to frequencies exhibited in historical data, hinting that his model provides considerable insight into how these empirical data are generated in their considerably diverse environments.

Note that it is easy to know that version I of Simon's model is simply a special case of his version III model. Since if the parameter γ is set to a constant 1, the weighted usage of each individual record is equivalent to its accumulated number of accesses. Thus the probability of it being accessed in the next selection process is proportional to its total number of accesses so far, which is exactly the second assumptions in Simon's version I model.

2.4 Applications of Simon's Model

Simon's model provides us a constructive way of modeling the information accessing. Study has shown that

Simon's modeling process not only provides data observations that resemble empirical data, it can also explain all the skew distributions discussed above (Chen et al. 1994).

Kim (1990) applied Simon's model (first two versions) to the performance analysis of self-organizing heuristics. First analytically, using Simon's two basic assumptions of version I model, he provided an explicit expression for the bound of the expected cost of MTF to be:

$$1/t\ [c_1\alpha^{t-1} + c_2\alpha^{t-2}(1-\alpha) + c_2\alpha^{t-3}(1-\alpha)^2, \ldots, c_2\alpha(1-\alpha)^{t-2} + c_t(1-\alpha)^{t-1}]$$

where $c_1 = t(t+1)/2$,

$c_2 = 1/2\ (t-1)\{t2-(t-2)/2\}$,

$c_t = t$, and

α is the constant entry rate of "new" record,

t is the number of access so far.

He also derived the expected search cost of the optimal static ordering under a more realistic formulation of Mendelbrot's law of word frequency which was derived by Chen and Leimkuhler (1990). However, he failed to give an explicit expression for the relative performance of MTF through analytical examination.

Simulation study was also performed to study the self-organizing heuristics by applying Simon's earlier version models in generating the usage frequencies (Kim 1990). The results from Kim's study are generally consistent with that

in the literature. The application of Simon's modeling process relaxes the unrealistic assumptions of record accessing frequency distribution. However, Simon's first two versions of model have limitations in terms of modeling the diverse frequency distributions in the real word. The main reason is due to the lack of ability to adjust the degree of autocorrelation. As a result, the application of Simon's version III model is imperative in our study.

2.5 Open Problems

Although there has been a significant amount of research on the performance study of database buffer replacement algorithms and self-organizing heuristics, the works in these two fields are generally unrelated and several problems remain unsolved at the present stage. These include unrealistic assumptions in analysis, the optimal replacement algorithms and heuristics, and locality modeling.

2.5.1 Unrealistic Assumptions

Most previous comprehensive performance models for database systems have used the simple assumption of uniformly distributed accesses. This assumption limited the ability of those models to reflect the performance improvement achievable through buffering (Casas and Sevcik 1989). Another commonly used model is one where each record has a fixed reference probability and where the references

are mutually independent. In other words, string of database references is modeled as a sequence of i.i.d. random variables, which is known as the independent reference model (IRM). While the IRM model implies independent database accesses, the accesses need not be uniform over all pages, which is a relaxation of the uniform assumption.

As a result, the importance of skewed data access for database applications is discussed in the literature (Casas and Sevcik 1989). The need to relax the independent assumption in accessing sequence has also been proposed (Berg and Towsley 1993).

Similarly, one primary problem with the majority of research on the performance analysis of self-organizing heuristics lies in their unrealistic assumptions. These include: (1) assuming incoming search requests are independent of each other; (2) assuming the probability of access to each data record is fixed over the whole search sequence and follows certain empirical distribution. It is well known (Hester and Hirschberg 1985) that incoming requests depend on the previous searches performed and the access probabilities are skewed and being dynamically changed at least over some short time period. Thus, performance evaluation under more reasonable assumptions which allows the dependent phenomena as well as changing

probability of access to each record would be more realistic and more valuable.

2.5.2 Optimizing the Algorithms and Heuristics

According to Sacco and Schkolnick (1986), no single strategy can guarantee a good replacement for all types of applications. Kaplan (1980) has shown that the buffer access fault rate can be cut 10-15 percent by replacement policies specifically chosen for the given type of evaluation (Stonebraker 1981). This fact motivates us to design some replacement algorithms that are more flexible in terms of fitting the application environment.

On the self-organizing heuristics side, recent study has centered on the discovery of an optimal heuristics over classes of probability distributions and heuristics. Rivest (1976) defined an optimal heuristics to be the one with the least cost for all probability distributions and any initial list order. He showed that TR has a lower asymptotic cost than MTF and also conjectured that this result extends to all heuristics (Rivest 1976). However, Anderson et al. (1982) provided a complex counterexample to Rivest's conjecture by presenting a single permutation algorithm that outperforms TR in the average case for a specific access probability distribution. This counterexample disproved the superiority of TR heuristics.

A few hybrids were suggested to combine the best features of various heuristics, such as using MTF initially and then switch to TR (Bitner 1979). However, it is believed to be very difficult so far to find out the best switching time, for example, from MTF to TR, not to mention choosing the exact time point based on analytical study results.

Further study under more reasonable assumptions is necessary in order to answer the question of optimal buffer replacement algorithms and heuristics. Specifically, we need to know under what conditions, a certain replacement algorithm is the best among the available candidates. It is very important to study and identify the procedure of deciding the optimal as the procedure must be not only reliable, but also feasible in order to implement.

2.5.3 Modeling the Locality

For buffer replacement algorithms study, one of the most important properties of a reference string is the locality of access. Locality means that the probability of accessing to the recently referenced pages is significantly different from the average accessing probability (Hester and Hirschberg 1985). It is a common and important attribute of access sequences. For example, in Pascal programs, predefined word "integer" tends to appear at the beginning of a source program file; and "end" tends to cluster

together at the end of a procedure or program. Similar phenomena can be found in many other situations.

Notice that the existence of locality is an important factor to consider when choosing among different heuristics. Researchers (Bently and McGeoch 1985; Bellow 1987) have shown that MTF performs better than TR if there is a strong temporal locality in a request sequence. Bently and McGeoch (1985), however, just provided the empirical results without any models which can describe the locality phenomena. Bellow (1987) attempted to model the locality in a search sequence by applying a discrete auto-regressive process, but he failed to relax the assumption of static probability of access in entire access sequence.

As we reviewed earlier, it is commonly assumed that the probability of access to each record is fixed (static) in any given search sequence, and accesses are also independent of each other. The average case analyses tend to assume that all access sequence (which has the same set of access probabilities) are equally likely. This assumption is obviously not valid in real situations due to the existence of locality. As taking the advantages of locality is one of the main reasons for using buffer replacement algorithms and self-organizing heuristics in linear search, it is important to model this phenomenon and only then the performance analysis would be valuable.

Though attempts have been made to modeling the locality phenomenon in access frequencies (Bellow 1987), as yet, no good approach capturing the locality of accesses has been successfully applied to the problem of performance analysis (Hester and Hirschberg 1985). New approaches are obviously desirable in order to get more accurate results.

CHAPTER 3 EXAMINATION OF SELF-ORGANIZING HEURISTICS

In this chapter, the general form of Simon's autoregressive model is applied in studying the performance of self-organizing heuristics. We first study the patterns of access frequencies generated from Simon's model. Then the model is applied to performance evaluation of self-organizing heuristics. As a result, tighter bounds of relative performance are obtained.

3.1 The Need for Computational Experimentation

Certain analytical study of Simon's model of information accessing has been conducted previously (Kim 1990, Chong 1994). Although researchers have shown that Simon's model of usage provides a unifying model; there are limitations in what analytical studies can do. Since conventional analytical methods can only derive the "average behavior" of the distribution, Leimkuhler (1988) suggested that computational experimentation be used for modeling empirical phenomena, especially in their applications to information system design and problem solving. He and other researchers argue that computational experimentation enables us to study the distributions under "extreme condition" and allows us to go beyond the analytical methods to examine details when the assumptions are relaxed (Leimkuhler 1988,

Neuts 1986). This is especially true when stochastic models admitting serial correlation have proved to be too complex to be solved explicitly in closed form for the equilibrium distribution (Ijiri and Simon 1977).

On the other side, computational examination has been popular in the literature of self-organizing heuristics. Although analytical studies of some self-organizing heuristics are possible under some simplified assumptions, simulation experimentation has always been used to study the performance of the heuristics under more complex access models. The incorporation of Simon's Version III model makes the simulation experimentation imperative in our study, partly due to the complexity of the dynamic behavior of Simon's model of information accessing.

3.2 Simulation Set Up

Our simulation data were obtained from programs written in VS Pascal Release 2. The basic system used was IBM mainframe under VM/CMS. Because of the significant amount of CPU time needed, the jobs were also executed in batch mode under MVS/ESA. Data from running the VS Pascal programs were downloaded into spreadsheet file and were used to generate graphs and tables.

Since stage two of the algorithm of Simon's Version III model requires that some "used" records exist, we assign an initial condition that there are three records in a list at

the beginning and each initial record has the same probability of being accessed in the next selection process (all three have weighted sum of accesses equal to 1 initially). The initial condition of records can be easily changed in our simulation program. This allows us to study the same phenomenon under different initial settings. However, our computational experimentation had shown, in general, similar results under different initial conditions. This is consistent with Simon's original observations. Therefore, the general conclusion from our initial examination again is that moderate changes in the initial conditions of list status do not appear to affect the equilibrium status (Simon and Van Wormer 1963).

To reduce the variance of simulation results, different seed values of random number generator can be selected during each run. In other words, the mean and variance of a performance measurement, e.g., cost, buffer miss ratio, under the input of different random number sequences can be calculated. This allows us to have a better idea of the range of values of a stochastic variable. Another method applied was using the result from a long simulation run instead of the mean of several short runs. As a result, the observations will be less affected by the initial condition selected as well as the random number sequence used. In this dissertation, we adopt the single long run method for

the most part of our study and use the first method whenever we deem necessary.

Using a long simulation run, an equilibrium status will reach eventually. The decision of the simulation run length was made by observing the change of values of our performance measurements obtained in test runs. It also depends on the setting of other parameters tested (e.g., list length, N, examined) as will be shown in the remaining of this chapter.

3.3 Examining the Accessing Frequency Distributions

3.3.1 Assumptions of Parameters

Simon's model of version III provides a more realistic modeling of access frequencies in real world by allowing the dependent phenomenon among subsequent accesses. It is also more flexible in modeling different access frequencies by allowing us to manipulate model's parameters. For example, in terms of Simon's version III model with autocorrelation, the second step in selecting a "used" record depends on the weighted sum of accesses of each individual record, which is influenced by the parameter γ (the decay factor). This more general model allows us to have a more flexible and versatile way of modeling the accessing frequencies.

One critical factor in applying Simon's version III model is the choice of parameters $\alpha(k)$ and γ. Simon explained in the book (Ijiri and Simon 1977) that $\alpha(k)$ is a

critical factor in his model and should be greater than zero all the times. However, in certain circumstances, it is possible that all selections will end up with "used" records, which means $\alpha(k)$ equals to zero under these situations. For example, at certain time point, all the data records have been accessed at least once (no "new" records any more) and all the following accesses could only be to "used" records. This is meaningful in some real information accessing cases since normally, we expect an information system will experience a time period where the process of information retrieval is stabilized and the only (or at least the majority of) operations are querying or updating, but no more "new" records being added.

Obviously, Simon's original models need to be adjusted in terms of the functional form of $\alpha(k)$ and the choice of γ in order to apply to some real applications. In our application case, the adjustments first need to be justified before they can be applied to study self-organizing heuristics. For this purpose, we need go back to Simon's original assumptions and study the changes necessary for our study.

As the first step towards studying self-organizing heuristics in Simon's model, we need examine the accessing frequencies distributions. For this purpose, we specifically define that $\alpha(k)$ is a linear function that

decreases a certain amount whenever a "new" record is being accessed. Assuming the maximum number of records in a list (i.e., list size) is N, the formal expression of our linear function is: $\alpha(k)=1-n(k-1)/N$, where $k=1,2,\ldots, R$, $n(k-1)=0, 1, 2,\ldots, N$, is the number of "used" records in the list after k-1 accesses, and R is the assumed maximum number of access.

In order to compare the accessing distributions generated from Simon's model here with those skew distributions applied in the literature, we further assume that the record accessing frequency distribution has the similar form of $P=(p_1(k), p_2(k), \ldots, p_N(k))$, where $p_1(k) \geq p_2(k) \geq \ldots \geq p_N(k) > 0$, $p_i(k)$ is the probability of access to the *i*-th record (in descending order of probability), and N is the number of records in a list. Note that under Simon's model, $p_i(k)$ is a function of k, the number of accesses so far. Compared with the ones in Section 2.2.3.1, $p_i(k)$ is no longer a constant but is dynamically changed as access proceeds.

Based on Version III of Simon's model, $p_i(k)$ can be generated from record i's weighted sum of accesses as follows. Let $w_t(k,i)$ be the weighted sum of accesses of record i at the end of k-th access, and W(k) be the sum of all "used" records' weighted sum of accesses, i.e.

$W(k) = \sum_{i-1}^{n(k)} w_t(k,i)$, where $n(k)$ is the number of "used" records in the list after k accesses, then we know $p_i(k)$, the probability of the i-th "used" record being accessed next, is $p_i(k) = \frac{w_t(k,i)}{W(k)}$, for $i=1,2,\ldots,n(k)$. When we take into consideration of the potential "new" records after k-th access (if there exist any), the above probability of the i-th "used" record being accessed should be $p_i(k) = (1-\alpha(k)) \times \frac{w_t(k,i)}{W(k)}$.

Assuming each "new" record has the same probability of being accessed, the probability of accessing to the j-th "new" records can be represented as: $p_j(k) = \frac{\alpha(k)}{N-n(k)}$, where $j=n(k)+1, n(k)+2, \ldots, N$.

In fact, in Version III of Simon's model of information accessing, the weighted sum of accesses associated with each record decides the potential of records being accessed. By converting the weighted sum of accesses to the probability of future access, a similar representation as in the definitions of the 80/20 rule and empirical skew distributions in Section 2.3.1, we are now able to examine the access frequency distributions graphically. It provides us a convenient method of comparing our results with those reported in the literature.

3.3.2 Simulation Results

To examine the access frequency distributions, we first assume N, the maximum number of records in a list, is 100;

γ, the decaying factor in Version III of Simon's model, be a constant of 0.99, and again $\alpha(k)$ has the linear decreasing functional form as defined in Section 3.3.1. Figure 3 displays the relationship between the cumulative probability of access to the number of records counted, where R, the number of accesses ranges from 100 to 500 with 100 interval. It is easy to find out that as R, the number of accesses, increases, the curves move towards northwest, which indicates that the access probability becomes more skewed. If we sort the records according to their probabilities of being accessed and observe the first 20 records that have higher probabilities of being accessed alone, we can see from Figure 3 that these 20 records have a cumulative probability of access of about 0.35 after 100 accesses are made; and this cumulative probability of access then increases to approximately 0.84 after 500 accesses are made. This observation clearly implies that as the number of accesses increases, the probability of access to the first 20 records that have higher access probability increases significantly.

The increasing trend in skewness of accessing will continue when more accesses are made, as shown in Figure 4, where R is up to 5000 in this presentation. The bold horizontal line in Figure 4 indicates that the majority of the 100 records are having zero probabilities of accessing,

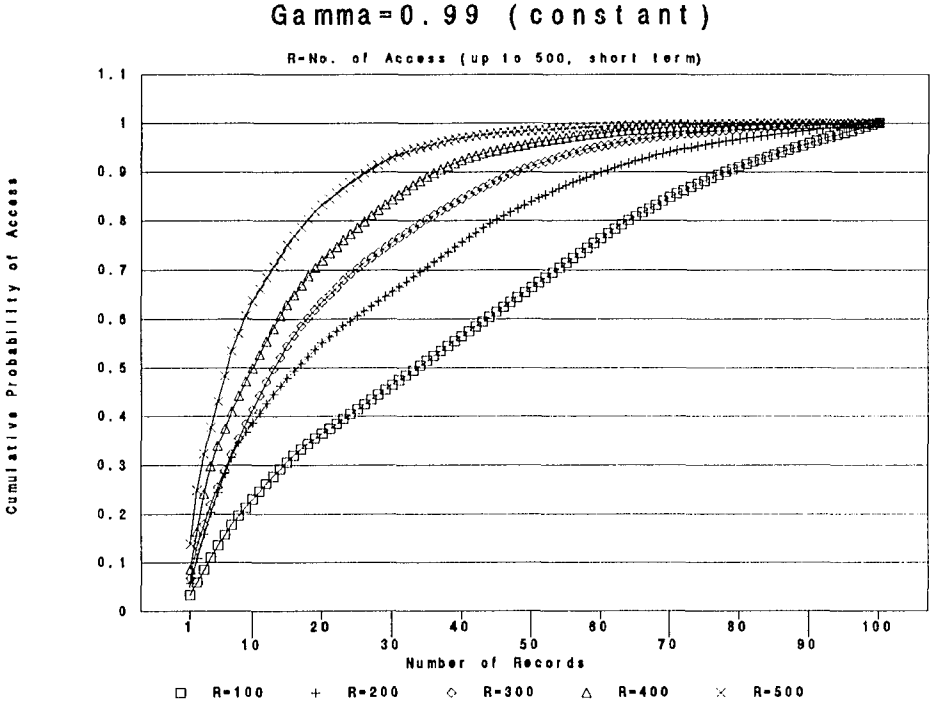

Figure 3: Short Term with γ = 0.99

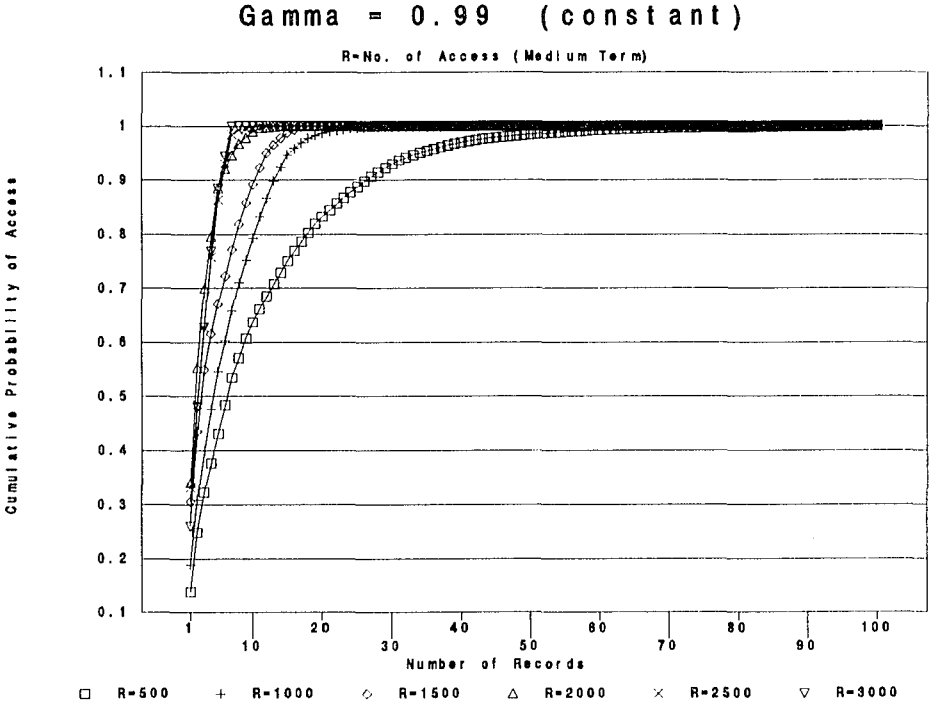

Figure 4: Medium Term with γ = 0.99

given R is larger than 1000. This zero probabilities of accessing can also be observed in Table 1 where both individual probabilities of access as well as cumulative probabilities of access are listed for each record in descending order of their values. In other words, these records with zero probabilities of access will never be accessed again when more accessing requests arrive. In fact, as the number of access further increases, eventually, there will be only one record with the probability of access one and all other records in the list have zero probability of access.

The general conclusion from observing both Figure 3 and Figure 4 is that as long as γ is less than 1, the skewness of access frequency distribution will increase as the number of accesses increases. The parameter γ will essentially decide the speed of increasing in skewness. The extreme situation of this skewness development is when all but one record has zero probability of access and thus all the subsequent accesses will be directed to the single record that has probability of access of one. This development of extreme situation is a special case that needs to be studied separately.

To avoid the extreme case above which rarely happens in the real world and generate a more realistic frequencies of accesses that are comparable with that in the literature,

Table 1: Record's Probability of Access (Sorted in Descending Order)

i	R=2,000		R=2,500		R=3,000	
	p_i	$\sum p_i$	p_i	$\sum p_i$	p_i	$\sum p_i$
1	0.34101	0.34101	0.33244	0.33244	0.26052	0.26052
2	0.21263	0.55364	0.15322	0.48566	0.22021	0.48073
3	0.14575	0.69938	0.14259	0.62825	0.14743	0.62816
4	0.09606	0.79545	0.12994	0.75819	0.14020	0.76836
5	0.08934	0.88479	0.10596	0.86414	0.11540	0.88376
6	0.03592	0.92071	0.06681	0.93095	0.05997	0.94373
7	0.02569	0.94640	0.05363	0.98458	0.05571	0.99944
8	0.02134	0.96774	0.00841	0.99299	0.00047	0.99991
9	0.01162	0.97936	0.00311	0.99609	0.00006	0.99996
10	0.01047	0.98983	0.00242	0.99852	0.00002	0.99998
11	0.00706	0.99689	0.00143	0.99994	0.00002	1.00000
12	0.00152	0.99841	0.00005	0.99999	0.00000	1.00000
13	0.00123	0.99963	0.00001	1.00000	.....	.....
14	0.00020	0.99983	0.00000	1.00000	.....	.....
15	0.00008	0.99992	.....	.....	.....	.....
16	0.00004	0.99996	.....	.....	.....	.....
17	0.00004	1.00000	.....	.....	.....	.....
18	0.00000	1.00000	.....	.....	.....	.....
.	.....	.....	.....	.....	.....	.....
.	.....	.....	.....	.....	.....	.....
100	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000

with all $p_i>0$, we need to reconsider the choice of parameter γ. Since for every $\gamma<1$, it will eventually result in the same extreme distribution. We thus consider the need to reset γ to 1 at certain time point. One case we studied is resetting γ to 1 when $\alpha(k)$ becomes zero. Figure 5 shows the simulation result under this adjustment, where R, the number of accesses, is again from 100 to 500 with 100 increment. It is obvious from observing Figure 5 that as γ is reset to 1 when α becomes 0, the trend of increasing of skewness is much slower than a constant $\gamma<1$ case as compared with that in Figures 3 and Figure 4.

To find out the long term effect of adjustment to γ, more simulation runs are carried out. Figure 6 displays the trend of probability of access distribution development as the number of accesses increases from 500 to 3000 with 500 increment. It is not difficult to find out that as the number of accesses R reaches a certain large number, for example, 2500, the distribution of the probability of accesses tends to be stabilized. This result is explainable because when $\alpha(k)$ becomes zero, all the records are "used" ones. Since the probability of access to each "used" record is proportional to its weighted sum of accesses, given γ is reset to 1, the changes in the weighted sum of accesses for all records will be proportional in general. As a result,

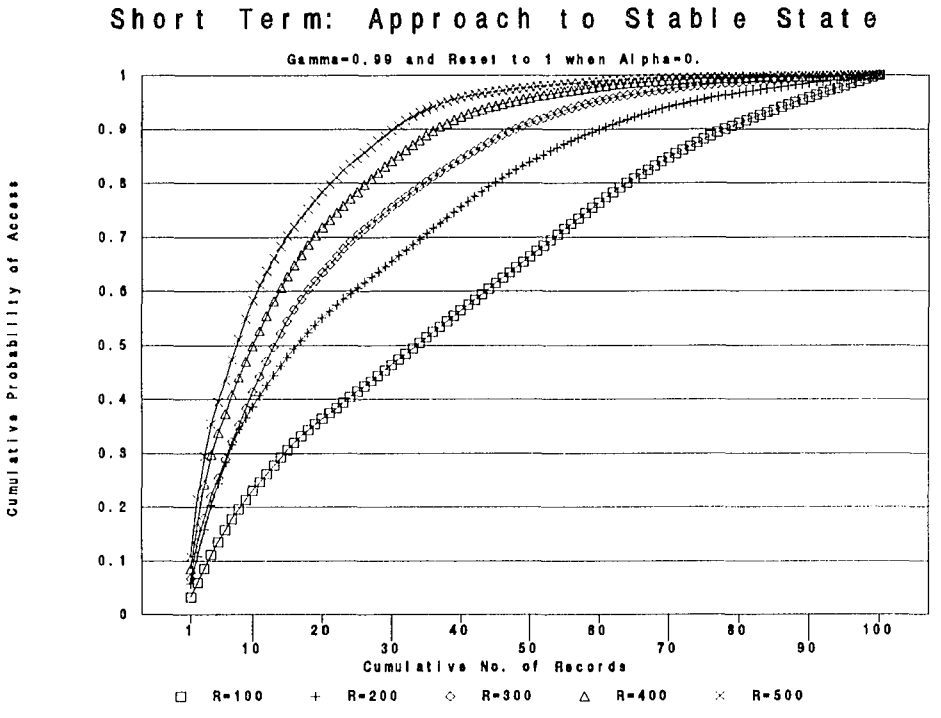

Figure 5: Short Term with γ = 0.99 and Reset to 1 Later

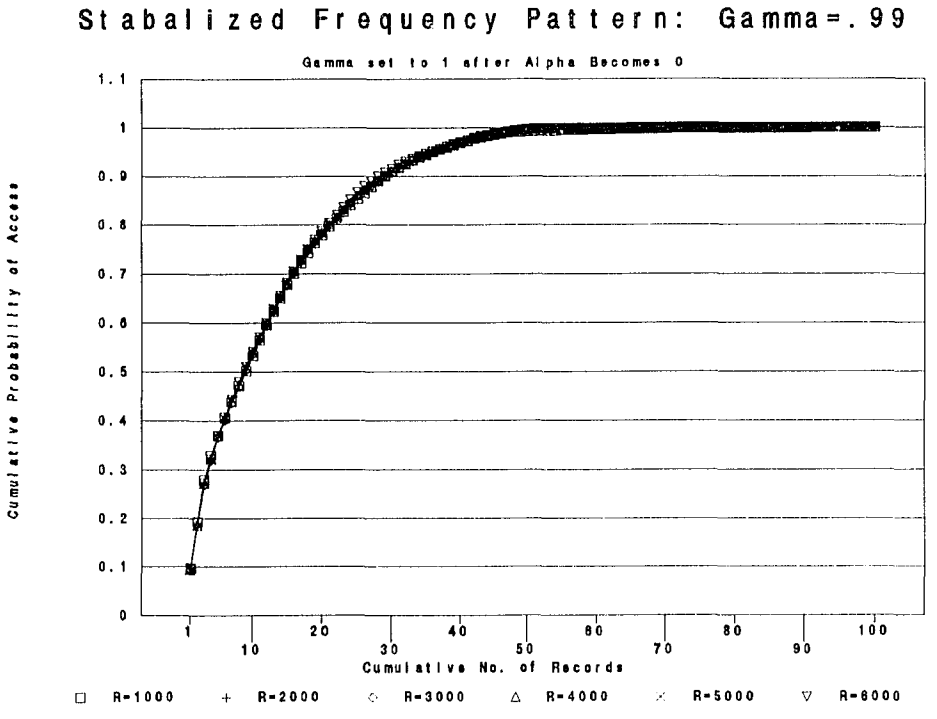

Figure 6: Comparison of Different R with γ = 0.99

this will avoid any significant changes in access frequency distribution and thus will not incur more skewness.

Different initial values of γ will result in different stabilized skew distributions. It thus gives different shapes of curves which describe the difference of skewness of accessing frequency distributions. Figure 7 compares the skewness of accessing frequency distributions where γ has initial value of 0.985, 0.99, 0.995, 0.999, respectively, and R, the number of accesses, is 3000 for all γ. Notice that given R equal to 3000, all the access frequency distributions have been stabilized in general, as it is observed from Figure 6. The observation from Figure 7 shows that as γ increases, the curves move towards southeast which means the skewness of the probability of future accessing decreases.

Note that when γ is initially set be 0.99, the corresponding curve in Figure 7 approximately gives a concentration measure of the 80/20 rule. This result comes from the fact that the curve shows that 20 records have a cumulant of 0.8 probability of access. In other words, 20 percent of records have a total probability of 0.8 of being accessed. This phenomenon is equivalent to the definition of the 80/20 rule in Section 2.3.1.1.

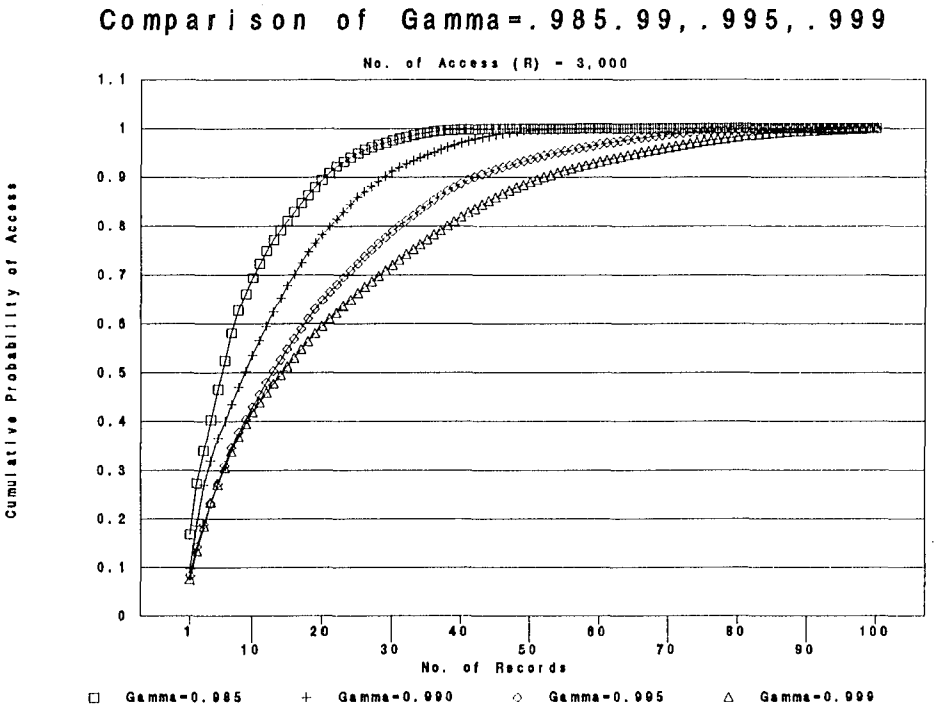

Figure 7: Comparison of Various γ

3.4 Performance Analysis of Self-organizing Heuristics

To study and compare the performance of various self-organizing heuristics, we adopt the average cost, which is equivalent to average number of probes per access, as an estimate of performance measurement. Throughout the simulation process, the program calculates and keeps records' average costs for different self-organizing heuristics, under an access sequence generated from Simon's model. Our study focuses on some basic types of heuristics, i.e., MTF, TR, A_k (k=1, 5, 10), Count, as well as their relative performance to the optimal static ordering.

Figure 8 shows some of the preliminary results from simulation experiment. The average costs of different heuristics under an access sequence that is generated from the adjusted version III of Simon's model are plotted in a line graph. The results are obtained by assigning initial γ be 0.99 and the number of access R from 1,000 up to a maximum of 50,000 with 1000 increment.

Some observations from Figure 8 are: (1) Count has about the same average cost as the optimal static ordering, and the cost difference between these two heuristics decreases as the number of accesses increases; (2) Transpose (TR, or in our experiment, A_1) has an initial significantly higher average cost. However, this cost decreases consistently as search proceeds. In contrary,

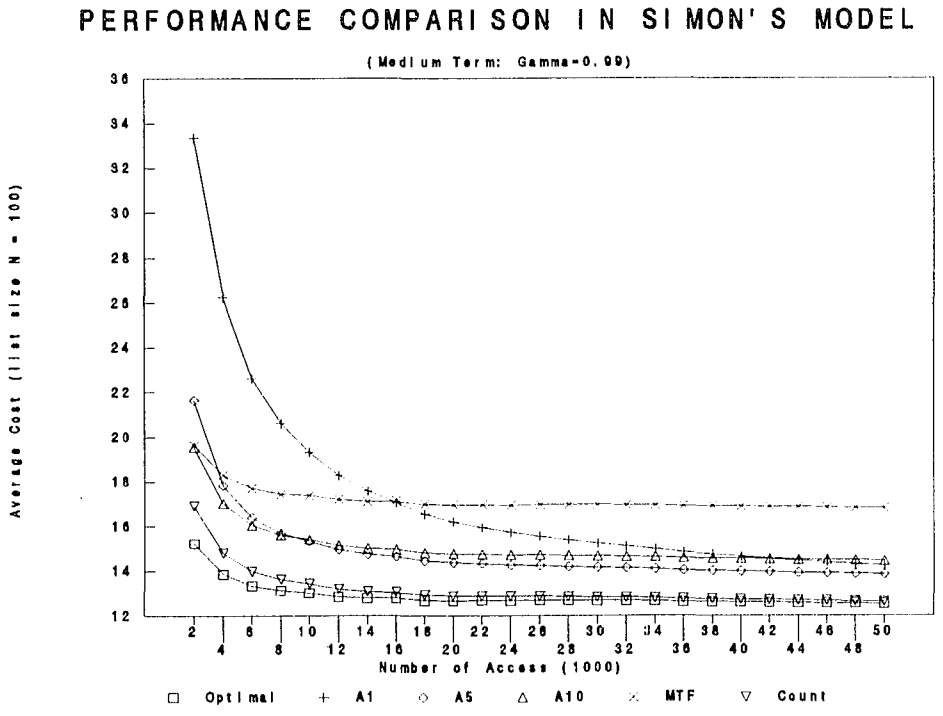

Figure 8: Heuristics Performance Comparison (Medium Term)

Move-to-Front has an initial relatively low average cost, but the long term average cost, i.e., the asymptotic cost, is greater than any of the heuristics shown in Figure 8. In other words, the asymptotic costs of A_1, A_5, and A_{10} are all significantly lower than that of MTF; (3) observation (2) can be generalized as, for i>j, the asymptotic cost of A_i is larger than A_j. In addition, A_i converges faster than A_j, i.e., A_i has a faster rate of convergence; and finally (4) the optimal heuristics depends on the number of accesses made, or, the timing of the process.

As more accesses are made, Figure 9 shows that the average cost of TR (or A_1) will be less than A_5, A_{10}, and MTF. It happens when R, the number of accesses, is greater than 48,000 approximately. Note that in Figure 8, when R is 50,000, TR (A_1) still has a higher average cost than A_5. Also, the lines of cost flatten out generally, especially for larger R, which means the heuristics are stabilized as R reaches a certain number. In summary, the primarily results from observing Figure 8 and Figure 9 are again, in general, consistent with those reported in the literature.

To study the relative measurements of heuristics, we also compared the expected average cost of MTF, TR, A_5, A_{10}, and Count with that of the optimal static ordering. The ratio or the bound of performance is summarized in Table 2. As Table 2 shows, when R, the number of accesses, increases,

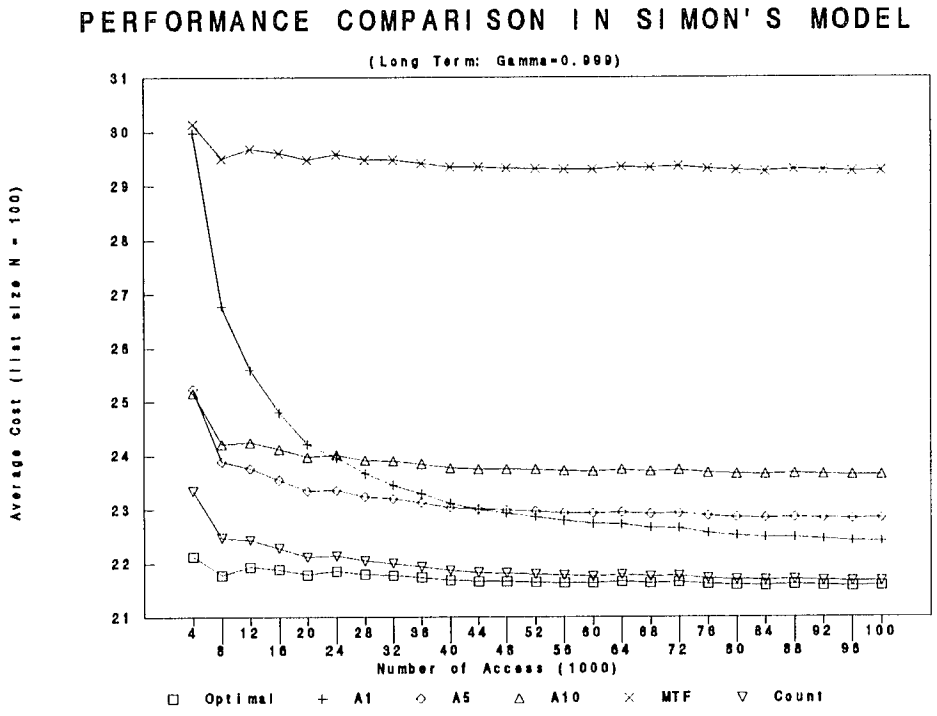

Figure 9: Heuristics Performance Comparison (Long Term)

Table 2: Relative Performance with $\gamma = 0.999$

R(1000)	$A_{TR}(P)$	$A_{AS}(P)$	$A_{A10}(P)$	$A_{MTF}(P))$	$A_{C}(P)$
2	1.50	1.23	1.18	1.34	1.10
4	1.35	1.14	1.14	1.36	1.05
6	1.28	1.11	1.12	1.36	1.04
8	1.23	1.10	1.11	1.35	1.03
10	1.19	1.09	1.11	1.35	1.03
12	1.17	1.08	1.10	1.35	1.02
16	1.13	1.08	1.10	1.35	1.02
20	1.11	1.07	1.10	1.35	1.02
24	1.10	1.07	1.10	1.35	1.01
28	1.09	1.07	1.10	1.35	1.01
32	1.08	1.07	1.10	1.35	1.01
36	1.07	1.06	1.10	1.35	1.01
40	1.07	1.06	1.10	1.35	1.01
44	1.06	1.06	1.10	1.35	1.01
48	1.06	1.06	1.10	1.35	1.01
52	1.06	1.06	1.10	1.35	1.01
56	1.05	1.06	1.10	1.35	1.01
60	1.05	1.06	1.10	1.35	1.01
64	1.05	1.06	1.10	1.35	1.01
68	1.05	1.06	1.10	1.35	1.01
72	1.05	1.06	1.10	1.36	1.00
76	1.04	1.06	1.10	1.36	1.00
80	1.04	1.06	1.10	1.36	1.00
84	1.04	1.06	1.10	1.36	1.00
88	1.04	1.06	1.10	1.36	1.00
92	1.04	1.06	1.10	1.36	1.00
96	1.04	1.06	1.10	1.36	1.00
100	1.04	1.06	1.10	1.36	1.00

the ratios of asymptotic cost under our study to that of the Optimal Static Ordering generally become stabilized, with the asymptotic cost of MTF and Count start stable first. On the other hand, Transpose takes much longer time to stabilize due to its conservative permutation rule. If we compare the ratios among the heuristics studied, it is easy to find out that when R is small, e.g., R=2000, the performance of the heuristics A_k is from A_{10} to A_5 to A_1 (TR) with A_{10} having the smallest ratio. When R=100,000, the performance is in order of A_1 to A_5 to A_{10} with A_1 the best.

In summary, the relative performance measurement we observed from simulation experiments are again consistent with that in the literature. Furthermore, the bounds we obtained through Simon's model of information accessing are significantly less than that from analytical study. For example, the ratio of $A_{MTF}(P)/A_O(P)$ is about 1.36 in our experiment, while the ratio obtained analytically by Chung (Chung et al. 1988) is $\frac{\pi}{2}$ ($\approx$1.57). This means our bound is tighter than that from analytical study, or in other words, the performance of self-organizing heuristics could be much better than analytical study results show.

3.5 Summary of Findings

The general observations from our simulation study are that the results we obtained are consistent with those reported in the literature. Furthermore, the bounds of

relative performance we obtained are tighter. This indicates that the self-organizing heuristics works even better than the analytical study result in the literature promised. As it is reported, the result of performance measurement heavily depends on the assumptions of accessing frequencies. Our primarily results are therefore justified since successive accesses generated from Simon's model are not independent in our study. We believe by applying our more realistic model of information accessing, the tighter bound obtained is more reliable and thus more approximate to what happens in real applications.

In addition, certain advantages of using our adjusted Version III of Simon's model were identified and can be studied further. These are:

1) By applying the adjusted Version III of Simon's model, we are able to model various usage frequencies under different environments. This is realized by estimating and manipulating the model parameters' form or values. In fact, the flexibility of this new approach allows us to apply Simon's model to virtually all suitable applications. In contrary, previous modeling approaches in the literature do not have these flexibilities;
2) Given Simon's innovative idea of modeling approach, parameters $\alpha(k)$ and γ are meaningful and explainable

under various application environments; this in term will help us identify and justify appropriate parameter values. For example, in studying self-organizing heuristics, $\alpha(k)$ means the probability of accessing to a "new" record, while γ describes how past accesses will affect the probability of the current access to a "used" record.

3) This same approach provides a potential means of more complex usage frequencies modeling. More complicated cases may be considered. For example: (a) the cases where $\alpha(k)$ is no longer constant or a monotonic decreasing function; (b) γ is not a constant during the entire stochastic process. These variations will not only allow the accessing frequency distributions generated to be dynamically changed in the short run (when $\alpha(k)>0$), as well as the existence of approximately stable frequency distribution in the long term (after γ is set to 1) as in our previous discussion, it will also allow various dynamic behaviors in any intermediate stages during the entire process of generating an access sequence.

CHAPTER 4 EXAMINATION OF BUFFER REPLACEMENT ALGORITHMS

In this chapter, we extend our study to database buffer replacement algorithms. Computational experimentation are first conducted on some traditional buffer replacement algorithms, as well as a class of replacement algorithms we proposed from studying self-organizing heuristics. Then, the performances of these algorithms are compared under various database access patterns and the findings of our study are summarized at the end of chapter.

4.1 Introduction

Similar to what we discussed in the last chapter, computational experimentation is frequently applied as a low cost feasibility study in designing a DBMS, and thus is very useful in studying the performance of buffer replacement algorithms. However, in order to carry out the computational experimentation under Simon's model of information accessing, we still need some assumptions and approximations. For example, we need first answer the following two questions: (1) What is an appropriate functional form or value for parameter $\alpha(k)$ in our study? and (2) What are the realistic assumptions in terms of the database buffer size and database size?

Regarding the first question, we have been assuming that parameter $\alpha(k)$ is constant and keep this consistent with Simon's original assumption of parameter γ in our study of buffer replacement algorithms. This assumption can be justified because of the following two reasons: (1) In real world database applications, accessing a database system can be generally assumed to be an infinite process. All the database accesses are essentially of three types: insertion, updating (read/write), and deletion. It is realistic to assume that there is a constant probability that an access to a database is an insertion type. In other words, the access is being made to a new record which needs to be inserted initially. This probability of accessing a new record (probability of new entry) is conceptually equivalent to the first assumption in Simon's model, which was discussed in section 2.3.2. Although there should always be limitation on the physical size of a database practically, the periodic operation of purging or archiving old data allows new data to be inserted at a reasonable rate; (2) There have been extensive research works regarding Simon's model of information accessing, with parameters α and γ being constants reported in the literature (Chen et al. 1993, 1994; Chong 1994). These studies not only justified the effects of parameters α and γ on the accessing frequencies generated, but also provided us a theoretical

foundation to select constant α and γ in studying database buffer replacement algorithms. Note that the exact value of parameters α and γ depends on the specific application environment where the model is applied, as we discussed previously.

As for the second question, our assumptions of database and buffer size are based on similar studies reported in the literature (O'Neil et al. 1993; Effelsbery and Haerder 1984). For example, O'Neil et al. (1993) assumed that the size of database buffer ranged from 40 to 500 pages, for a database of 1000 pages in their study. Effelsbery and Haerder (1984) indicated that buffer size varied from about 16 K to 12 M bytes and a typical buffer size might be assumed to be between 128 K and 256 K. Assuming a page is of 1024 bytes, it is thus reasonable to study the cases where a database buffer size is in the range of 10 to 500 pages. Furthermore, to study the performance of various replacement algorithms, Effelsberg and Haerder selected two database reference strings from a CODASYL DBMS where the schema and transactions were taken from a school database application. Each reference string accesses 3,553 and 5,245 different pages respectively. In other words, 3,553 or 5,245 accesses were made to new database pages. Assuming the total number of accesses in both reference strings is 30,000, it is reasonable to estimate that parameter α has an

approximately value of 3,553/30,000 or 5,245/30,000 respectively. Based on the above observations, we select the value of α mostly in the similar magnitude in our study.

Throughout the study in this chapter, we selected that the size of database buffer ranges from 10 to 200 pages, and database size ranges from 500 to 5,000 pages by selecting an appropriate α under a given number of accesses. The number of accesses used in our current study is 5,000. Although other assumptions of database and buffer size may be reasonable, due to the computational resources required in conducting our simulation study, we generally limited our assumptions of parameters to the above mentioned ranges.

The computational environment in this study is quite similar to that of last chapter. Our simulation program was designed to be general enough so that it could handle different database and buffer sizes, as well as various patterns of access frequencies by allowing us to change model parameters easily. The simulation program is driven by a database reference string generated from Simon's version III model, where parameters α and γ are specified as input. The output of the simulation program is the buffer miss ratio of various replacement algorithms as a function of buffer size and the database reference string. Various ranges of parameters were experimented in our study.

4.2 Performance of Buffer Replacement Algorithms

As an initial step towards identifying a database buffer replacement algorithm that is optimal for some particular applications, a preliminary performance evaluation of buffer replacement algorithms under Simon's model of information accessing were carried out through simulation study.

4.2.1 Optimal and Random Replacement

Although the Optimal replacement algorithm (OPT) is not practical, it is worth studying because it gives the theoretical lower bound for the buffer miss ratio of all algorithms, which is very important in performance evaluation of replacement algorithms. By assuming that an accessing string is generated from a certain model, we are able to predict precisely the future accesses and therefore to obtain the buffer miss ratio for the OPT algorithm.

Assuming a database reference string including 5,000 accesses, Figure 10 is a set of three curves describing buffer miss ratio as a function of buffer size. The parameter γ of accessing model is set to be 0.97, 0.98, and 0.99 respectively and α is fixed at 0.3. It is obvious from observing the graph that a larger γ corresponds to an initially higher miss ratio and also a slower convergence to the optimal miss ratio (the horizontal line in the graph), which is approximately the value of parameter α. In other

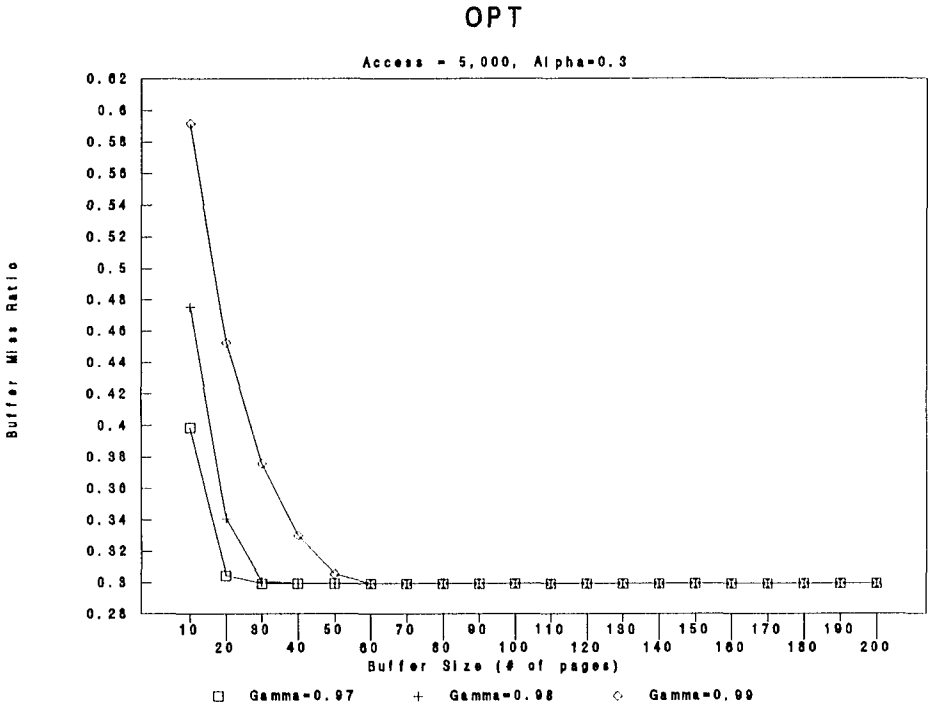

Figure 10: Optimal Replacement under α = 0.3

words, all others being the same, a reference string with a smaller γ needs less buffer space in order to reach the optimal (minimum) miss ratio. In the case of Figure 10, when γ is 0.97, the smallest buffer size needed is about 30 pages while if γ is 0.99, it will need a minimum of about 60 pages to obtain the same miss ratio.

As for the effect of parameter α on the result of buffer miss ratio, Figure 11 shows that α will mostly decide the values of the optimal miss ratio, and also with some effect on the minimum buffer size required to reach the optimal miss ratio. The larger the α, the larger the buffer is needed in order to reach the optimal (stable) miss ratio. This phenomenon is easy to understand since a larger α means a database increases at a higher speed, and thus it needs a larger buffer to accommodate the new pages inserted into a database. Note that since each insertion is counted as a buffer fault in our study, the value of α is the least possible miss ratio no matter what others are specified.

Although the "absolute" upper bound of buffer miss ratio can be derived by the algorithm which replaces the buffer page with the shortest access distance (contrary to Optimal replacement algorithm), a reasonable upper bound of buffer miss ratio for all practical replacement algorithms (algorithms that are designed properly) should be achieved by the Random Replacement (RR) algorithm, an algorithm that

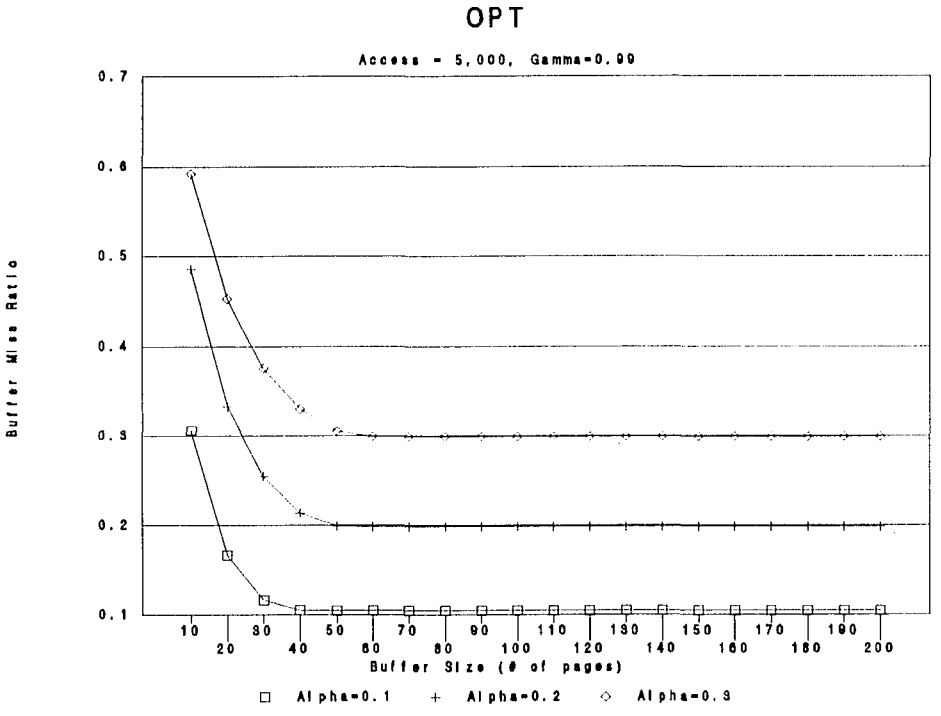

Figure 11: Optimal Replacement under γ = 0.99

does not use any knowledge about past reference behavior. Hence, for practical purpose, OPT and RR are assumed to limit the realm of reasonable algorithms, giving a quality measure and some hints concerning the optimization potential with respect to a given replacement algorithm (Effelsberg and Haerder 1984).

To study the random replacement algorithm, we first plot the curves of buffer miss ratio with α fixed but γ varies from 0.97 to 0.99 as shown in Figure 12, and similarly, curves with γ fixed but α varies from 0.1 to 0.3 in Figure 13. Compared with those in Figure 10 and Figure 11, the curves here again all have downward slopes, which means the buffer miss ratio decreases as buffer size increases. However, as we compare the curves associated with RR here with those associated with OPT, the curves here generally have flatter slopes. In other words, they converge slowly than those of OPT algorithm. Higher γ again incurs higher initial miss ratio, as well as relatively higher miss ratio even though buffer size increases. In fact, it seems that the stable horizontal lines of miss ratio for all three curves in Figure 12 do not converge to a single line. Instead, higher γ will result in higher miss ratio even when buffer size increases to 200 pages in our cases. Again, curves from higher γ converges slowly compared with curves from smaller γ. And finally, all

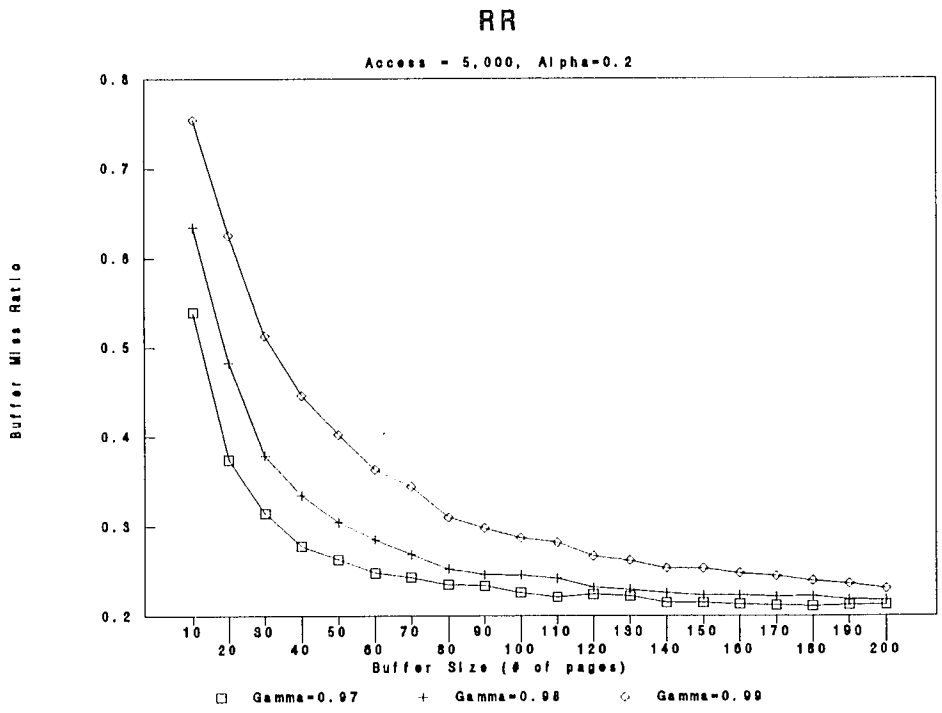

Figure 12: Random Replacement under α = 0.2

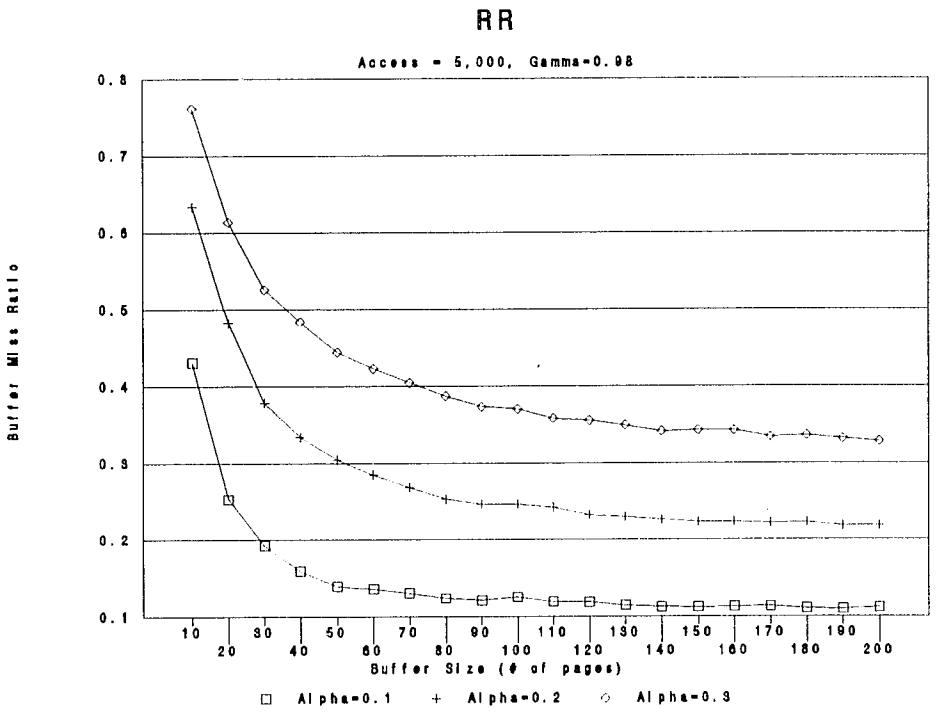

Figure 13: Random Replacement under γ = 0.98

stable optimal miss ratio are greater than α, which means the RR algorithm can never reach the Optimal miss ratio.

Figure 13 compares the cases when γ is fixed and α varies from 0.1 to 0.3. Again, the values of α will in general limit the lower range of stabilized miss ratio (the nearly horizontal part of the curves). While curves with a smaller α not only have a smaller stabilized miss ratio, it also converges faster (has steep slopes) as it can be observed from Figure 13.

4.2.2 First-In, First-Out

Among the various algorithms reported in the literature, one of the most widely studied algorithms is the First-In-First-Out (FIFO) algorithm. Figure 14 and Figure 15 are designed to study the effect of α and γ on the buffer miss ratio under FIFO algorithm, given that the number of accesses is 5000 and buffer size is from 10 to 200 pages.

The general patterns of curves under FIFO algorithm are similar to those under OPT and RR in the last section. With the value of α designating the stable miss ratio in general and γ affects the initial value of miss ratio when buffer is small in size. However, the FIFO algorithm requires relatively larger buffer compared with OPT algorithm in order to reach the stable miss ratio. Also, this stabilized miss ratio approaches the value of α, with curve of smaller

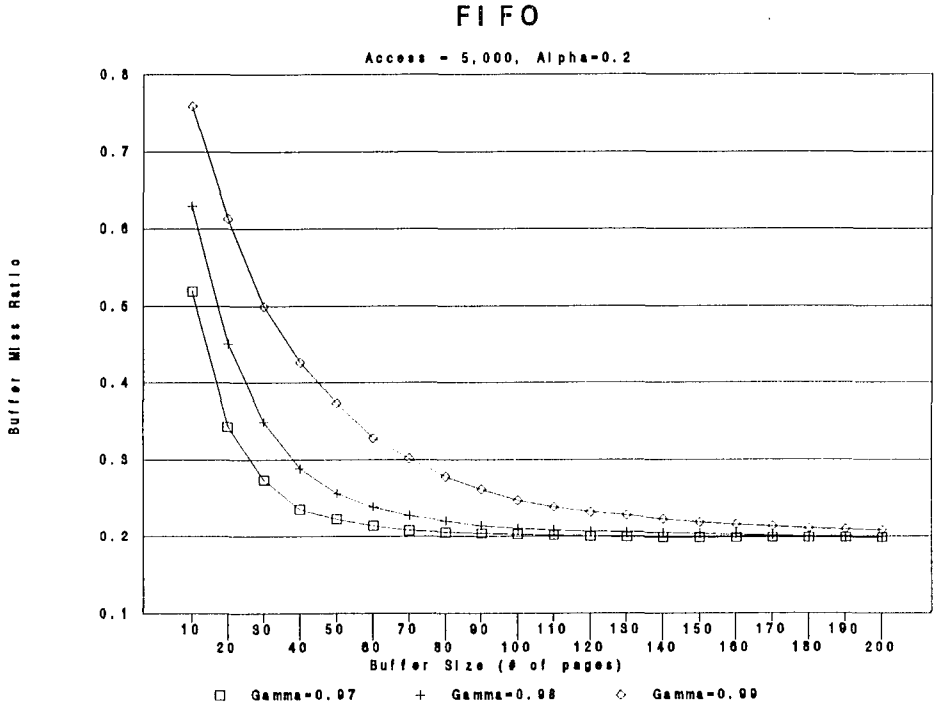

Figure 14: FIFO Replacement under α = 0.2

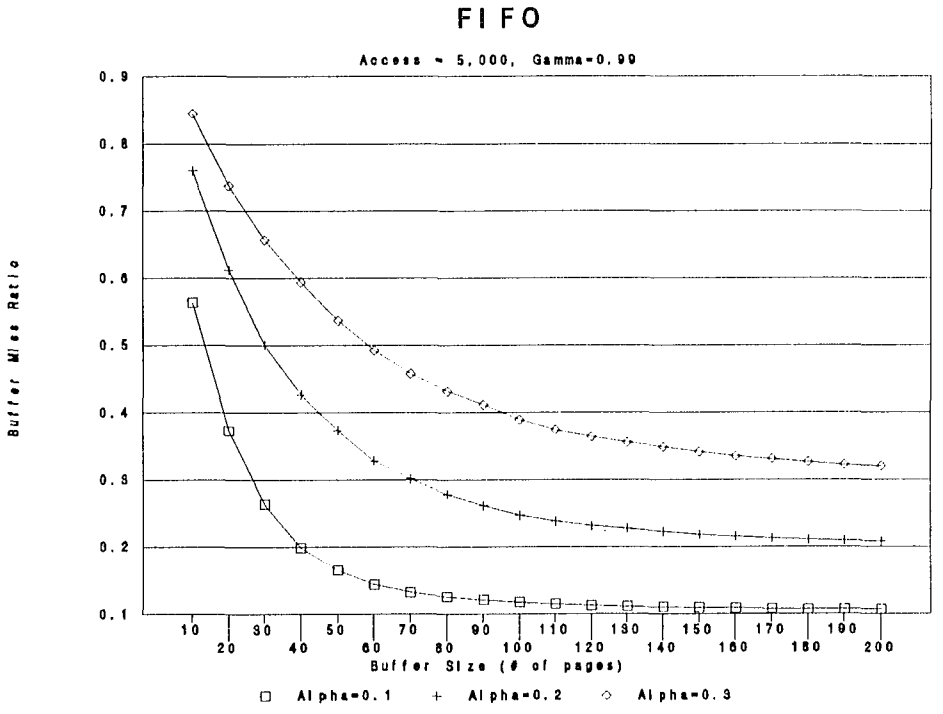

Figure 15: FIFO Replacement under $\gamma = 0.99$

γ converges faster than that of larger γ if they have same α, as in the case of Figure 14.

4.2.3 Least Frequently Used

If we think about a database buffer as if it were organized as a linear list, the Least Frequently Used (LFU) algorithm is conceptually equivalent to the Count heuristics studied in Chapter 3. Notice that records (pages) in buffer are not necessarily physically reorganized as what happened in self-organizing linear search. Instead, the implementation can be storing records' index in a linear list as illustrated in the examples in section 2.1.1. Only index in this linear list is reorganized as references to database are being processed.

Figure 16 and Figure 17 are the curves describing the buffer miss ratio when number of accesses is 5,000 and buffer size ranges from 10 to 200 pages. Besides the same characteristics identified in the previous sections, the figures here show that the LFU algorithm converges even slower though we have specified the same parameters' values here. In other words, the LFU algorithm requires much larger buffer in order to reach the optimal buffer miss ratio that is designated by parameter α. Also, Figure 17 shows that the effect of γ on the database buffer miss ratio is not as obvious as that in the previous algorithms because the three curves are close to each other. Figure 16 also

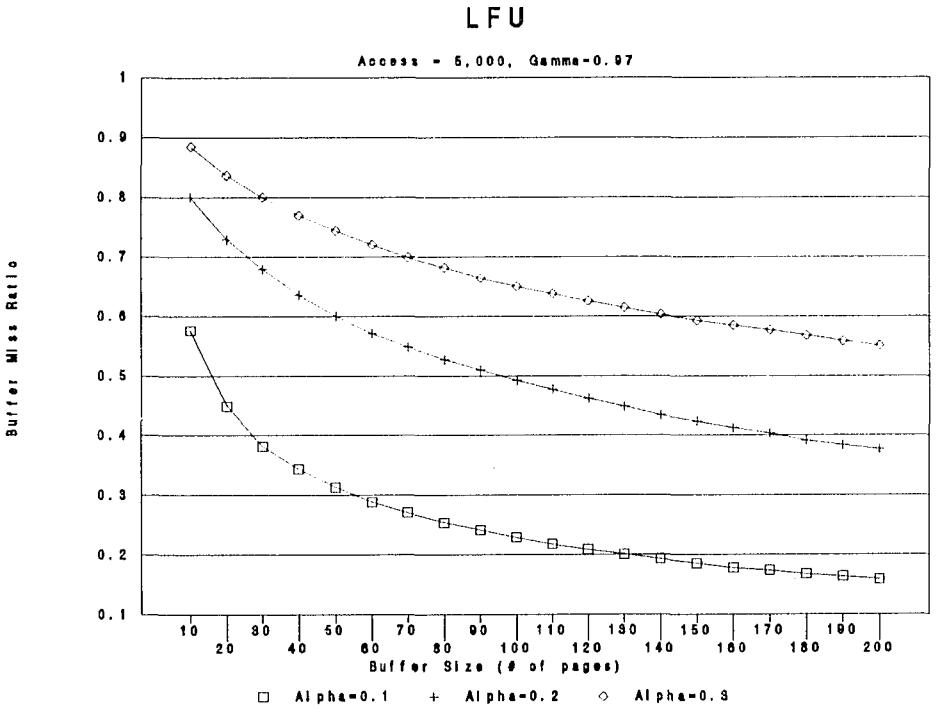

Figure 16: LFU Replacement under γ = 0.97

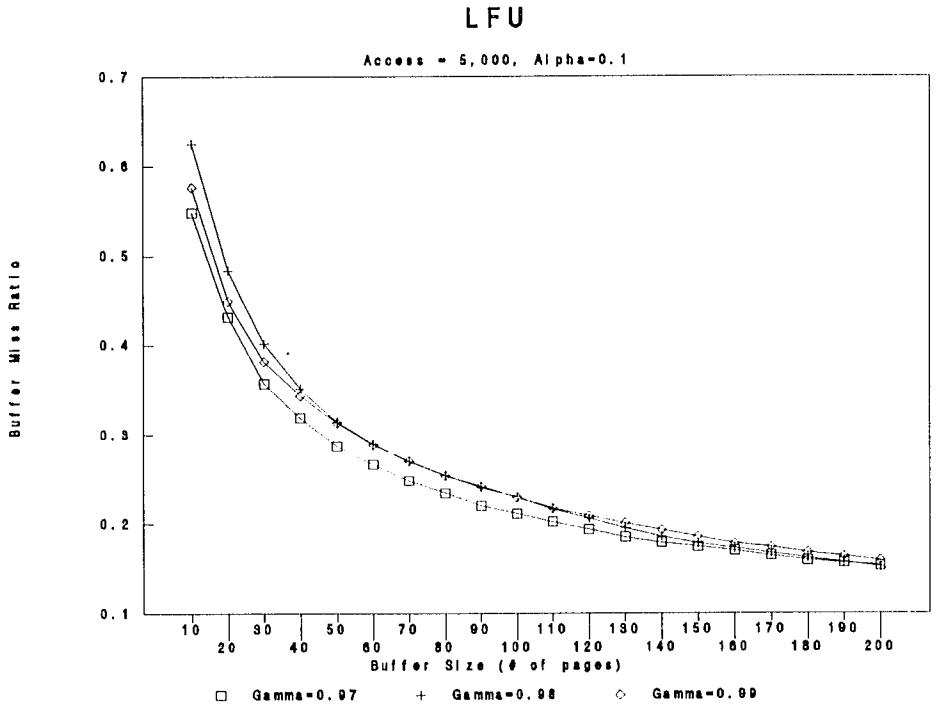

Figure 17: LFU Replacement under α = 0.1

shows that all others being the same, the buffer miss ratio obtained from the LFU algorithm is much higher than that of any of the previous algorithms, or in other words, far above the horizontal line of α.

4.2.4 Least Recently Used and Variations

Another extensively studied and applied replacement algorithm is the Least Recently Used (LRU) algorithm. Following the explanation in the last section, it is easy to know that the LRU algorithm is logically equivalent to the Move-To-Front heuristics in chapter 3. Therefore, each replacement will replace the page whose index in the self-organizing list is farthest from the front.

The patterns of curves in Figure 18 and Figure 19 are pretty much similar to that of the OPT algorithm, with exception that the LRU requires relatively larger buffer size in order to reach the stable miss ratio, especially when either α or γ is larger if all others stay the same. Here again, Figure 18 shows that a curve with a smaller γ converges faster than a curve with a larger γ, although they eventually reach the same ratio given that they have the same value of α. Figure 19 gives evidence that under the same γ, a curve corresponding to smaller α converges to a smaller stabilized miss ratio faster than a curve corresponding to a larger α. And again, the parameter α in general decides the stabilized miss ratio in all cases.

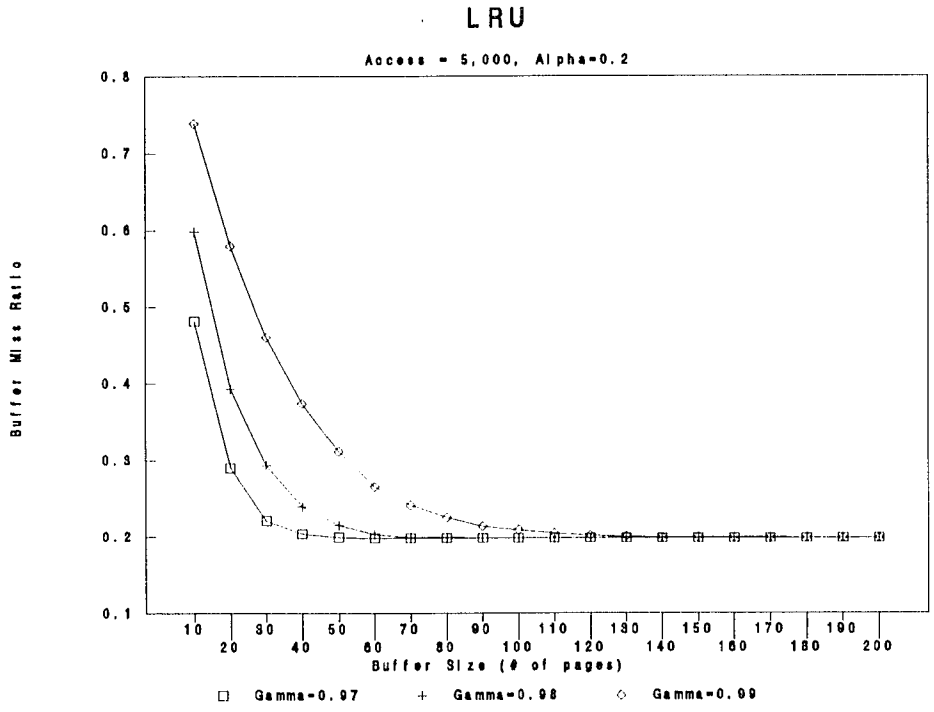

Figure 18: LRU Replacement under α = 0.2

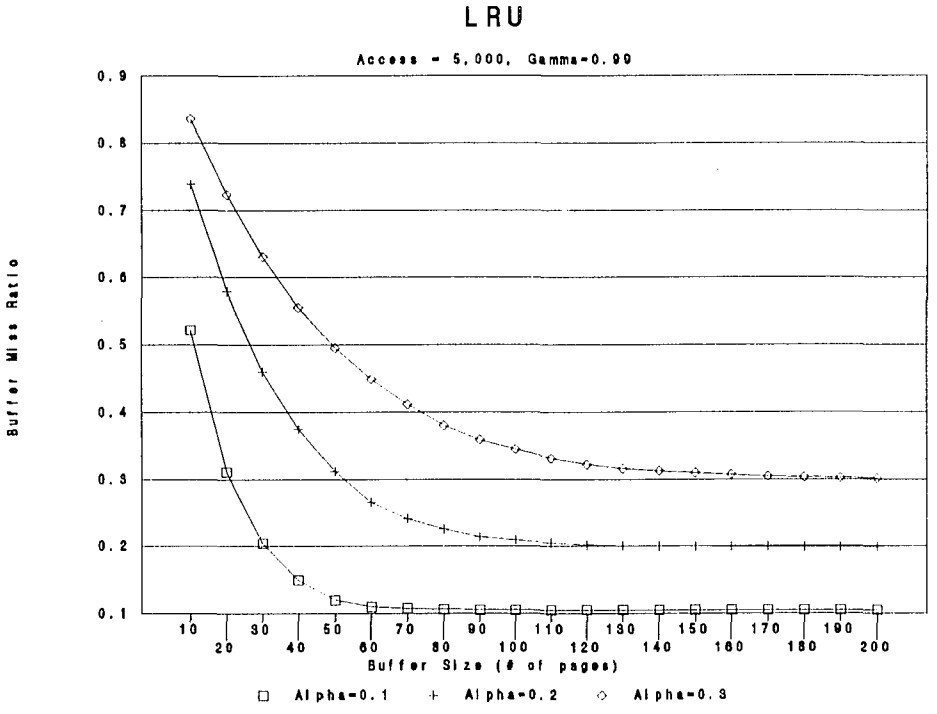

Figure 19: LRU Replacement under γ = 0.99

From the study of self-organizing heuristics in the last chapter, we already know that a generalization of the MTF heuristics is the so called Move-Ahead-K (A_K) heuristics. Although the buffer replacement algorithm corresponding to Transpose (TR or equivalently A_1) has been reported in the literature (Ch'ng et al. 1987, 1988), thus far, to our knowledge, there has been no report on the more general buffer replacement algorithms which correspond to the A_K heuristics. Since these replacement algorithms are essentially the same type, we will study them together and also use the corresponding names in self-organizing heuristics to denote their counterpart in buffer replacement algorithms.

The functional curves of miss ratio for replacement algorithms of this type with various K are summarized in Figure 20. With all others being the same, a curve corresponding to a smaller K generally has a higher miss ratio. However, this is not true when buffer size is especially small, where the observation is just the opposite, i.e., a curve with smaller K has a smaller miss ratio as shown in Figure 20. Also, A_K algorithms with larger K converge faster than those with smaller K. For instance, in Figure 20, algorithm A_{10} reaches an approximately stable miss ratio when buffer is about 90 pages, while the algorithm A_1 is far from being stable at

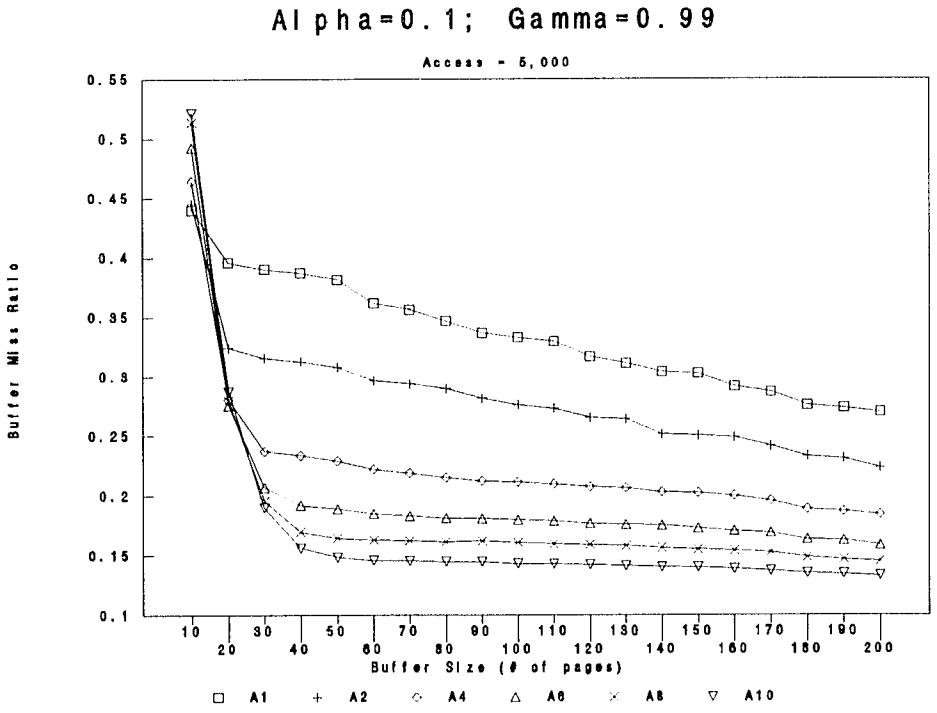

Figure 20: Comparison of A_K under α = 0.1 and γ = 0.99

this buffer size. This difference in the speed of convergence can also be observed by comparing the slopes of various curves in Figure 20.

In order to study the effect of parameters α and γ on the buffer miss ratio with all others being the same, we selected the algorithm which corresponds to A_5 as an example. The curves corresponding to various α and γ are classified into two groups as shown in Figure 21 and Figure 22 respectively. As we can see from Figure 21, the effect of γ on buffer miss ratio is more significant compared with previous replacement algorithms, with larger γ incurs larger buffer miss ratio if all others stay the same. Also, the buffer miss ratio curve converges to a horizontal line much slowly. For instances, given the buffer size ranges in Figure 21 and Figure 22, the buffer miss ratio is still away from convergence and far from the possible value that is designated by parameter α.

4.3 Comparisons of Replacement Algorithms

Given the number of database buffer replacement algorithms reported in the literature, it is important to compare the performance of various replacement algorithms so as to identify the optimal algorithm that is applicable under certain conditions.

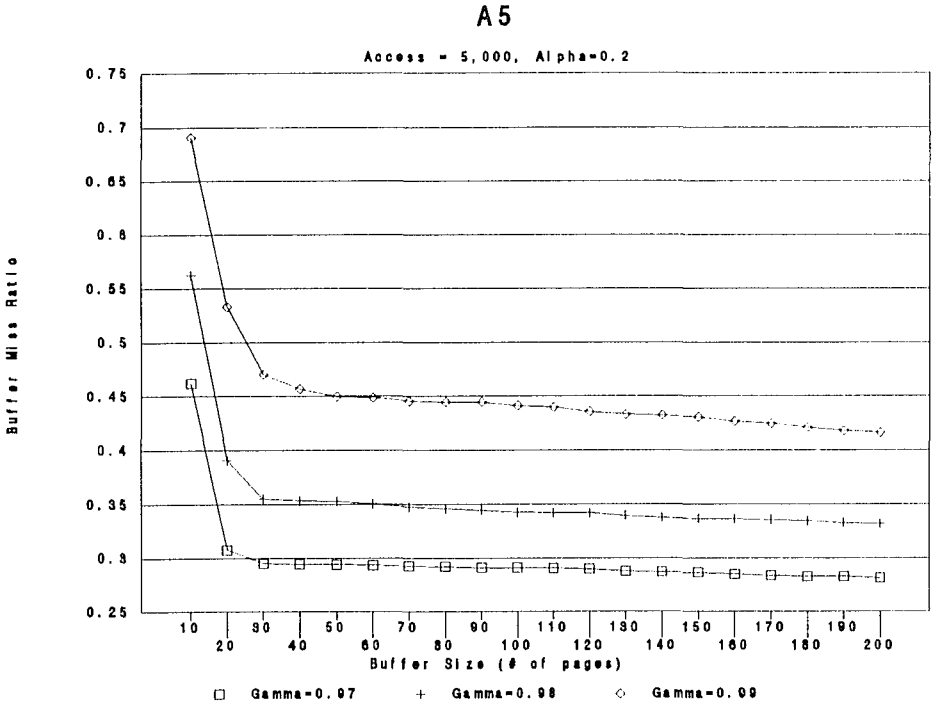

Figure 21: A_5 Replacement under α = 0.2

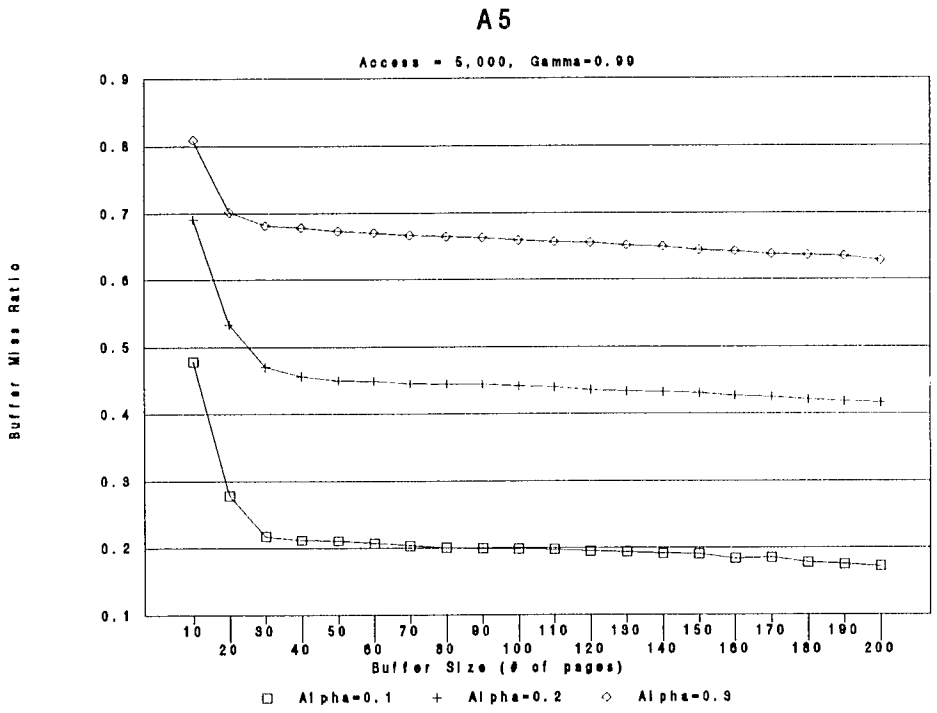

Figure 22: A_5 Replacement under $\gamma = 0.99$

4.3.1 Classification of Algorithms

Before we start computational experimentation on the relative performance of various buffer replacement algorithms, it is a good idea to analyze the difference among the various algorithms based on their definitions. The result of this will in general guide us in selecting what parameters to test, what to expect, and how to interpret the simulation results.

Replacement algorithms can generally be classified according to the way the database accesses and accessing age are taken into consideration. An overview of the previously discussed replacement algorithms is given in the table below:

CONSIDERATION DURING REPLACEMENT		**AGE**		
		No Consid-eration	Since Most Recent Access	Since First Access
ACCESSES	No Consider-ation	RANDOM		FIFO
	Most Recent Accesses		LRU	
	All Accesses	LFU		$\mathbf{A_K}$

The positions of various algorithms in this table are given by their definitions. Notice that the class of newly proposed algorithms A_K derived from studying the Move-Ahead-K heuristics has certain distinctive features compared with

other algorithms, in terms of its consideration of the accesses and age of accesses and thus is in a unique position in the table above. The A_K algorithm in fact considers all accesses and their age, while none of the other four algorithms listed in the able has these properties. Therefore, we expect that the A_K can not be replaced by other algorithms listed in the table.

4.3.2 Preliminary Performance Comparison

In order to compare the performance of various replacement algorithms, we have to initially select some sample parameter values since the large number of possible combinations of parameter values and replacement algorithms prohibits an exhaustive study. The result of initial study will generally provide us some ideas as what further study can be made in terms of parameter value ranges and algorithms comparison, etc. Unfortunately, the various algorithms have to be displayed in different graphs to avoid an overload representation, and at least allow for a clear separation of the inefficient algorithms. For example, Figure 23 combines the curves for some of the traditional buffer replacement algorithms with parameters α and γ set to 0.3 and 0.97 respectively.

Since the bounds of reasonable replacement algorithms are given by Random replacement and Optimal replacement algorithm respectively, a rough observation of Figure 23

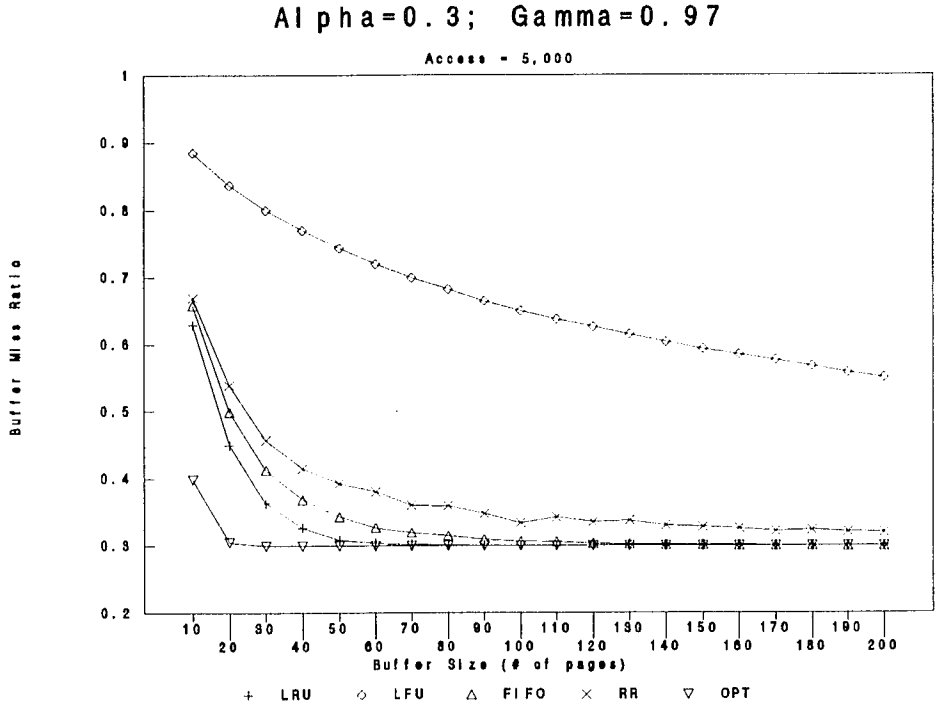

Figure 23: Performance Comparison under α = 0.3 and γ = 0.97

will come to a conclusion that the LFU algorithm is far from being reasonable and may be discarded from our study. However, the fact that LFU is not reasonable under a special reference string specified by a specific pair of α and γ here does not necessary mean that LFU algorithm is not reasonable in all circumstances. In fact, as we will see later on, in certain occasions, LFU is optimal among all the algorithms under our study. Other observations are that both LRU and FIFO algorithms have reasonable performances, and LRU outperforms FIFO under this setting. And finally, as buffer size increases, both FIFO and LRU approach the Optimal performance with LRU reaches the horizontal line first with much less buffer space required, and neither LFU, nor RR seem to approach the optimal horizontal line with some buffer size increasing.

Figure 24 represents the similar information without including the LFU algorithm so that the presentation is better. With the parameter α set to 0.1 in this diagram, the order of performance stays the same under all the given buffer sizes. Also, with α set to 0.1 here, much less buffer space is needed in order for the LRU to reach and converge to the possible minimum buffer miss ratio. On the other hand, the space required for FIFO to do the same stays about the same from comparing Figure 23 and Figure 24.

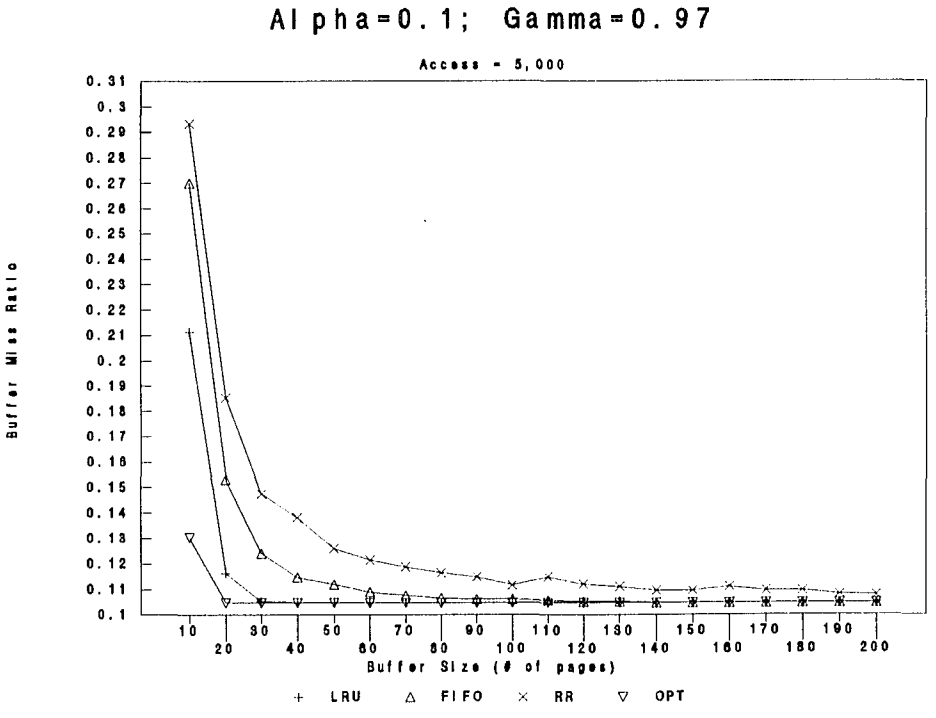

Figure 24: Performance Comparison under α = 0.1 and γ = 0.97

The performance of A_K algorithm with K equals a constant 10 is compared with that of FIFO algorithm in Figure 25. With parameter α being set to 0.1 and γ selected to be 0.97, A_{10} is generally within the bounds set by RR and OPT here. In addition, when the buffer is less than 60 pages in size, A_{10} actually outperforms FIFO by having a smaller buffer miss ratio. However, A_{10} is not as good as FIFO when buffer size increases over 60 pages, as we can easily see from Figure 25. In addition, A_{10} does not seem to converge to the optimal miss ratio as the FIFO algorithm does in Figure 25.

To find out more about the A_K algorithms with K of different values, we also include two more algorithms A_1 and A_5 in Figure 26. It is obvious that A_1 is well over the curve given by the RR algorithm and thus is unreasonable in this case. However, both A_5 and A_{10} are reasonable at least when the buffer size is limited. For example, A_5 is reasonable up till buffer size is approximately 30 pages and A_{10} is reasonable up till buffer size hits 60 pages. What is more interesting is that if we observe the relative performance of replacement algorithms when buffer size is 10, A_5 performs better than A_{10}. The general observation here is that the performance of A_K algorithm is very sensitive to the buffer size it applied to.

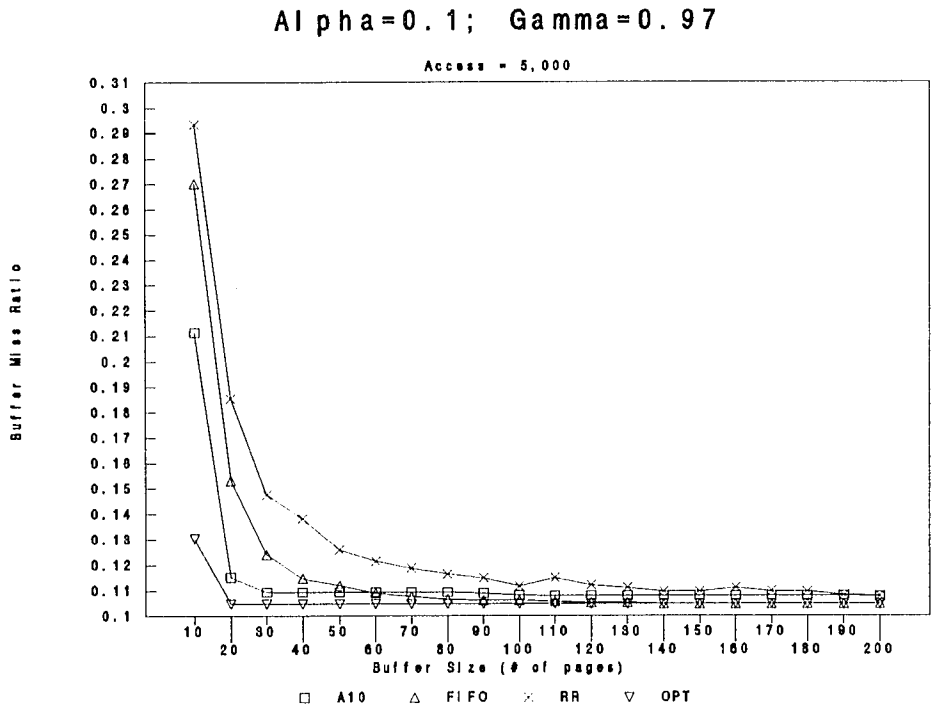

Figure 25: More Performance Comparison

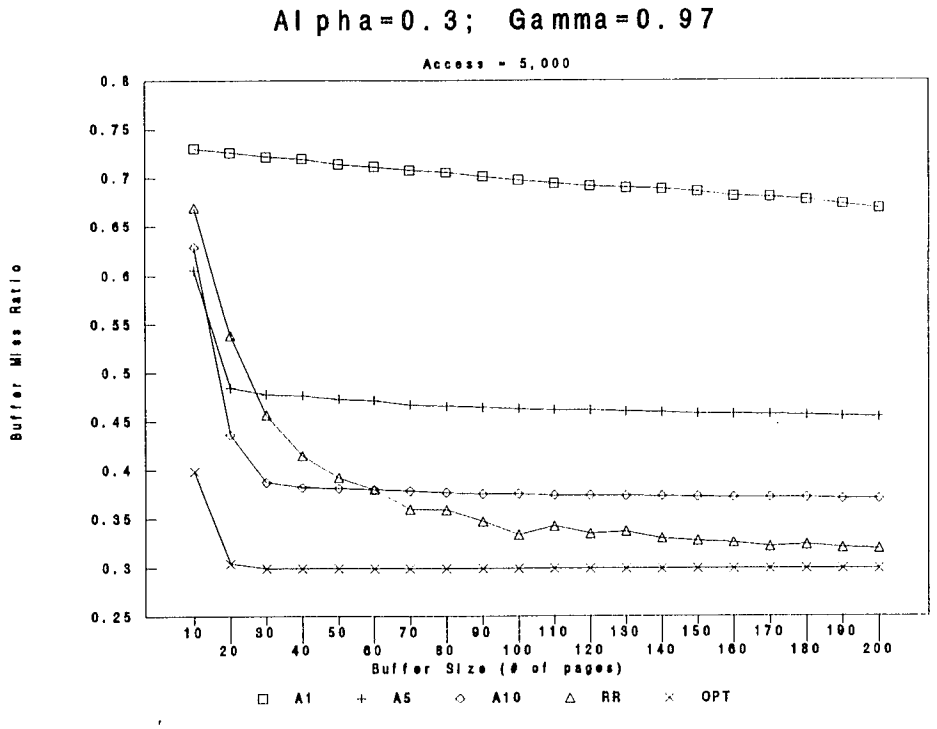

Figure 26: Performance Comparison under α = 0.3 and γ = 0.97

The performance of A_K algorithm with K equal to 5 and 10 respectively are further compared with that of the LRU algorithm, the one which so far has been shown to be superior among the algorithms studied. In Figure 27, one interesting finding is that when the buffer size is less than 30 pages, for example, 10 or 20 pages, there is always an A_K algorithm that outperforms the LRU algorithm. More specifically, when buffer size is around 10 pages, A_5 outperforms the LRU algorithm. On the other side, when buffer size is about 20 pages, Algorithm A_{10} performs better. The observations here hold true when α and γ is set to 0.3 and 0.97 respectively, thus, further investigations are needed in order to find out if similar results can be obtained for other parameters α and γ; and/or it is also true when K increases as the buffer size increases.

4.3.3 Optimizing the A_K Algorithm

Our observation so far from the previous comparisons is that among the algorithms FIFO and LRU, LRU generally outperforms FIFO. However, we have also shown that given certain reference strings, the newly proposed A_K algorithm with K set to a certain number and under certain buffer size may outperforms LRU. An immediate question is: Under what conditions, does this relation hold true?

As we find in Figure 27, A_5 has a smaller miss ratio than LRU when buffer size is 10 pages, and similarly, A_{10} has

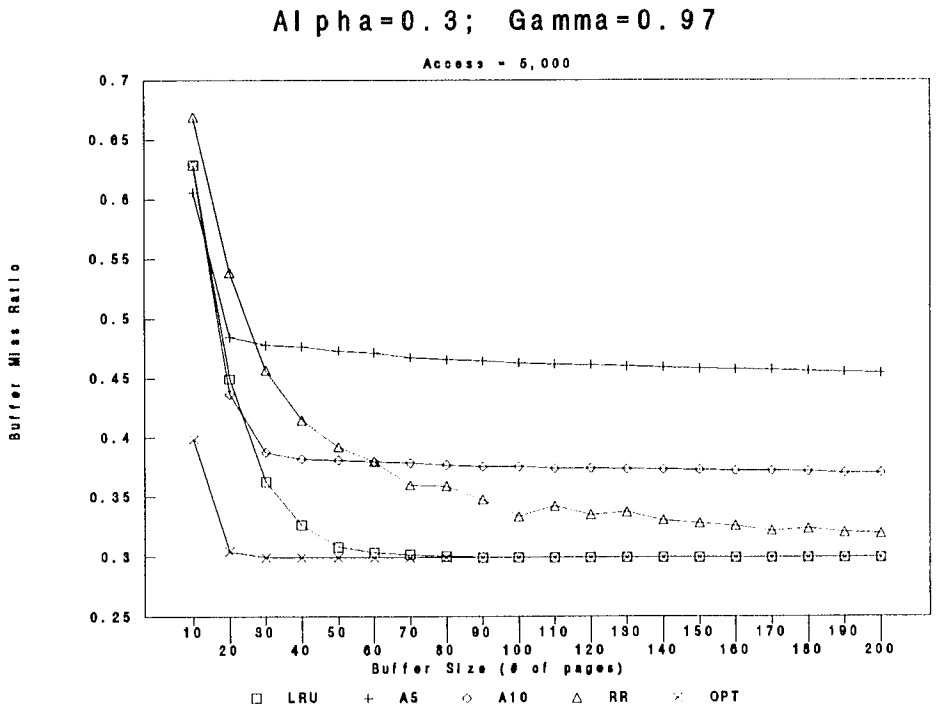

Figure 27: More Performance Comparison

a smaller miss ratio than LRU when buffer size is 20. It seems to us that A_K outperforms LRU only when K is assigned a number that is comparable with the buffer size, for example, K is set to be half of the buffer size. Figure 28 is then developed to study the A_K algorithm where K is set to be half the buffer size, and α, γ were assigned the same values as in Figure 27. However, the result in Figure 28 does not support our conjecture since A_K only outperforms LRU in the smaller range of buffer size, i.e., less than 30 pages. Although the difference of miss ratio between A_K and LRU in this diagram is significantly reduced compared with the cases where K has a maximum value of 10 before, and the miss ratio of A_K even approaches the same optimal value as buffer size increases, there is no consistency observed in terms of superiority between the A_K and the LRU algorithms in Figure 28.

Further studies were carried out by assuming different parameter values in Simon's information accessing model, which is equivalent to selecting different database reference strings. As we can see in Figure 29 and Figure 30, as γ approaches 1, A_K with K equals to half the buffer size consistently outperforms the LRU algorithm, at least within the range of system's set up here. Figure 29 was designed to study the effect of parameter γ and it is obvious that curves generated from reference strings

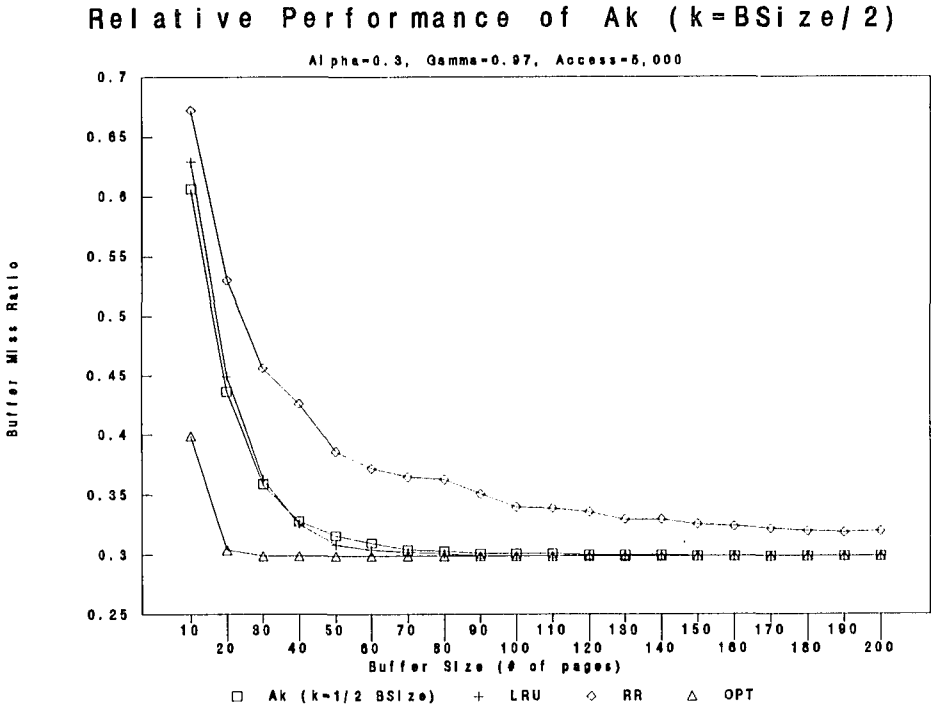

Figure 28: Performance of A_K with K = Half Buffer Size

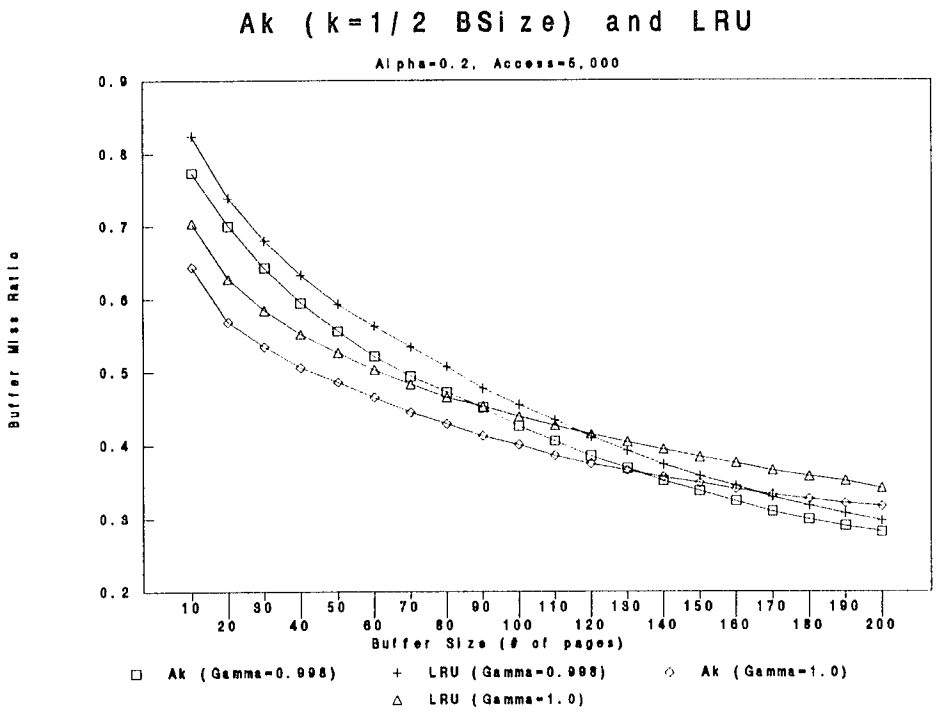

Figure 29: Comparison of A_K and LRU Replacement

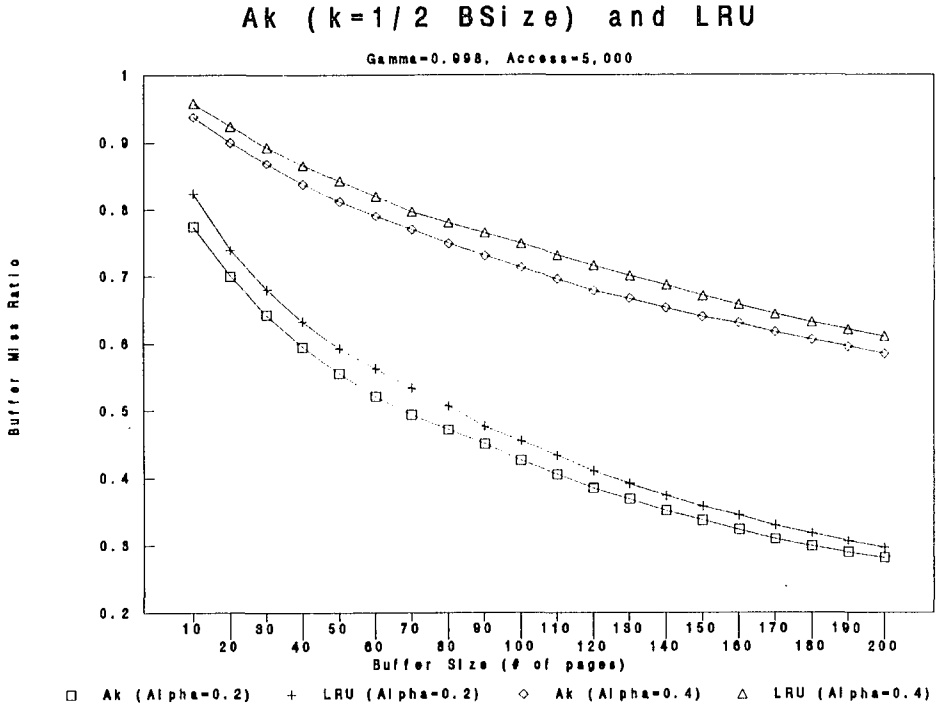

Figure 30: More Comparison of A_K and LRU Algorithm

associated with larger γ are flatter compared with that from smaller γ. In other words, the buffer miss ratio is relatively less sensitive to the increasing in buffer size when γ is relatively large. On the other side, if we set γ to be constant and observe the effect of α as in Figure 30, curves corresponding to larger α are flatter compared with curves corresponding to smaller α, i.e., the increasing in buffer size can decrease the buffer miss ratio more effectively if α is relatively small.

4.4 Summary of Findings

The general conclusions from our preliminary computational experimentation are: (1) buffer miss ratio normally decreases as buffer size increases; (2) the stabilized buffer miss ratio depends on the algorithm used and the value of parameter α, if the buffer size is reasonably large; (3) the sensitivity of buffer miss ratio to the buffer size depends on the parameter γ in most cases, and also may be influenced by parameter α in certain algorithms.

Among the traditional buffer replacement algorithms studied, LRU algorithm generally outperforms other reasonable algorithms, which could be one of the reasons it has been widely applied in DBMS. Also, the performance of FIFO is normally next to LRU, and LFU performs least efficiently in most cases. However, as we have found in the

previous study, this order of performance does not always stay the same. The relative performance of various replacement algorithms will also be affected by other factors such as buffer size, patterns of reference strings, etc.

We also proposed a group of algorithms from studying the Move-Ahead-K heuristics in linear search. Although the A_K algorithms we proposed do not display any performance advantages over the traditional algorithms when K is set to be significantly smaller than the buffer size, we have demonstrated that by setting K to approximately half the buffer size, this specific A_K algorithm will outperforms LRU under certain accessing frequency patterns. In fact, as our classification of replacement algorithms shows, the unique consideration of A_K algorithm will differentiate it from all other existing algorithms, and thus gives it performance advantages under certain situations.

CHAPTER 5 SELF-ADAPTIVE DATABASE BUFFER REPLACEMENT SCHEME

It is generally accepted that the effectiveness and efficiency of buffer replacement in a DBMS can significantly affect the overall performance of database systems and the computer on which it runs. Motivated by the findings in buffer replacement algorithms and self-organizing heuristics study, as well as the conceptual similarities between them, we propose a self-adaptive buffer replacement scheme in this chapter. This self-adaptive replacement scheme obviously has certain performance advantages over the traditional replacement algorithms.

5.1 The Need for Adaptive Replacement

As we know from the previous study, among the many buffer replacement algorithms studied, the most frequently used LRU algorithm generally has a better performance over other algorithms. However, LRU was developed originally for patterns of use in instruction logic (Coffman and Denning 1973) and does not always fit well into database environment, as was reported in the literature (Stonebroker 1981; Sacco and Schkolnick 1986). In fact, the LRU algorithm decides which page to be replaced based on too little information, limiting itself to only the time of the last reference. It was also reported recently (O'Neil et

al. 1993) that LRU is unable to differentiate between pages that have relatively frequent accesses and pages that have very infrequent accesses until the system has wasted a lot of resources keeping the infrequently accessed page in buffer for an extended period.

Obviously, there is a need to derive other buffer replacement algorithms that take into account more of the access history for each page, and to better discriminate pages that should be kept in buffer and those that should be kept in secondary storage. Subsequently, we proposed a class of buffer replacement algorithms that were originated from the A_K heuristics in self-organizing linear search. We have also shown that the general A_K algorithm with K set to be an appropriate value outperforms LRU algorithm, at least under certain reference strings.

However, with the diversity in database reference strings and also the various replacement algorithms tailored to specific characteristics of database reference behavior, it is then only conceivable to combine various replacement algorithms. This idea of combining various algorithms can also be justified because database accessing in real world applications is not always a static process, and given the apparent conflict in replacing the buffer page without considering different access patterns, there is a need for a buffer replacement scheme that is responsive to changes in

access patterns (Kearns and DeFazio 1989). Hence, further optimization of buffer replacement could only be obtained by assigning appropriate replacement algorithms based on a reference oriented scheme. This so called adaptive replacement should tailor its strategy to the accessing of database depending upon the observations of accessing patterns to avoid conflict and should be an important consideration for future applications.

5.2 Optimizing Replacement Algorithm

The selection of practical optimal buffer replacement algorithm depends on several factors of an application environment as well as the database system. Among these, the most critical factor to consider when selecting an appropriate replacement algorithm is the characteristics of the database reference strings. Since we propose that the reference string can be represented in Simon's model of information accessing, the characteristics of reference strings are essentially influenced by the values of model parameters α and γ.

The preliminary computational experimentation in the last chapter has in general indicated that the LRU algorithm outperforms other algorithms under most circumstances, with some exceptions depending on the values of parameters selected. In fact, further investigation shows that the LRU algorithm is not always optimal in all cases of database

references. There are certain cases where other replacement algorithms, especially the LFU and A_K algorithms, are the practical optimal algorithms among the algorithms we studied.

When we limit our consideration to the characteristics of reference strings, we are able to compare the performance of various replacement algorithms under the assumptions that the number of accesses is the same, as well as the range of database buffer size. Our objective is to identify the practical optimal replacement algorithm under various values of parameters α and γ.

The procedure we went through involves initially running simulation experiments under test parameter values and then fine tuning the values of the parameters in order to identify an optimal algorithm under that set of accesses. The trial and error procedure was applied since a comprehensive enumeration of parameter values is impractical given that both α and γ can practically be any real numbers between 0 and 1. Some important findings of our computational experimentation are summarized and displayed in Figures 31 through 35.

Figure 31 displays an obvious performance advantage of the LFU algorithm. Contrary to our initial observation in the last chapter, when α and γ are set to be 0.25 and 0.9999 (near 1), not only is the LRU algorithm outperformed by both

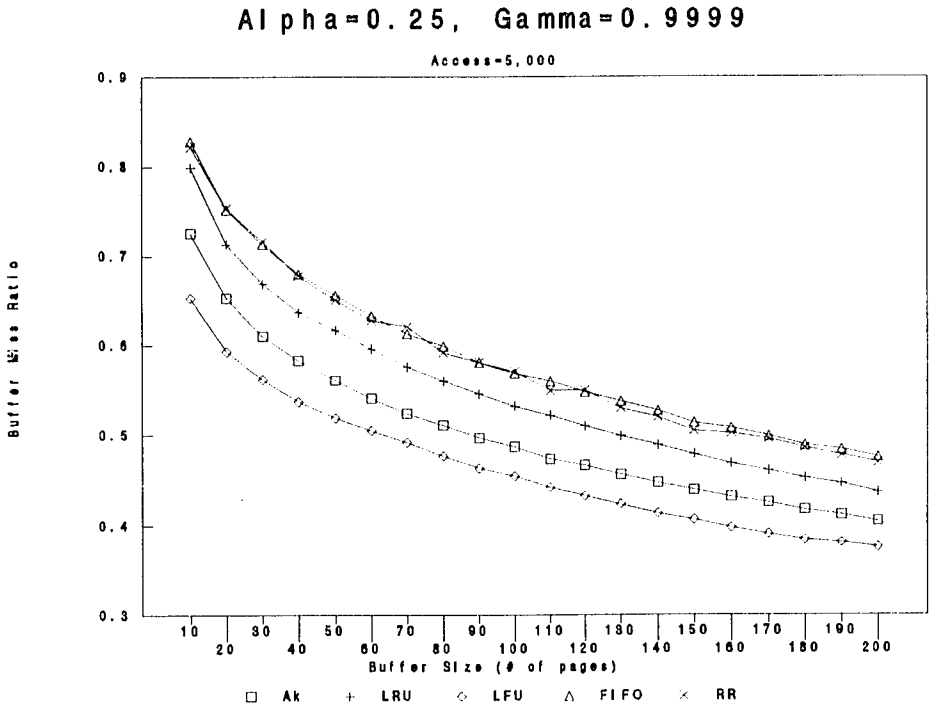

Figure 31: Relative Performance under α = 0.25, γ = 0.9999

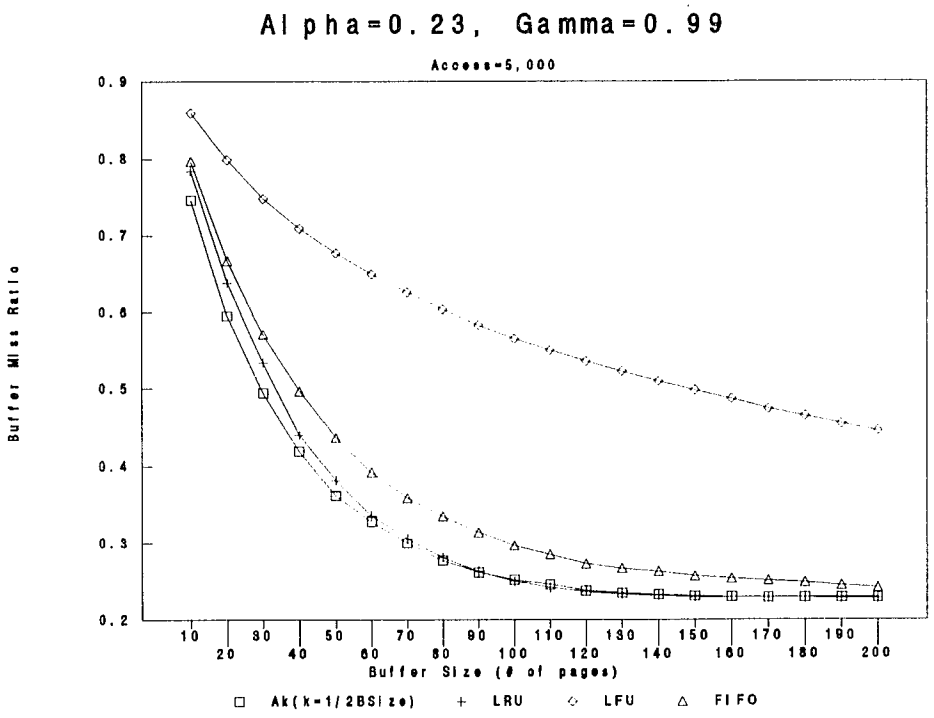

Figure 32: Relative Performance under α = 0.23, γ = 0.99

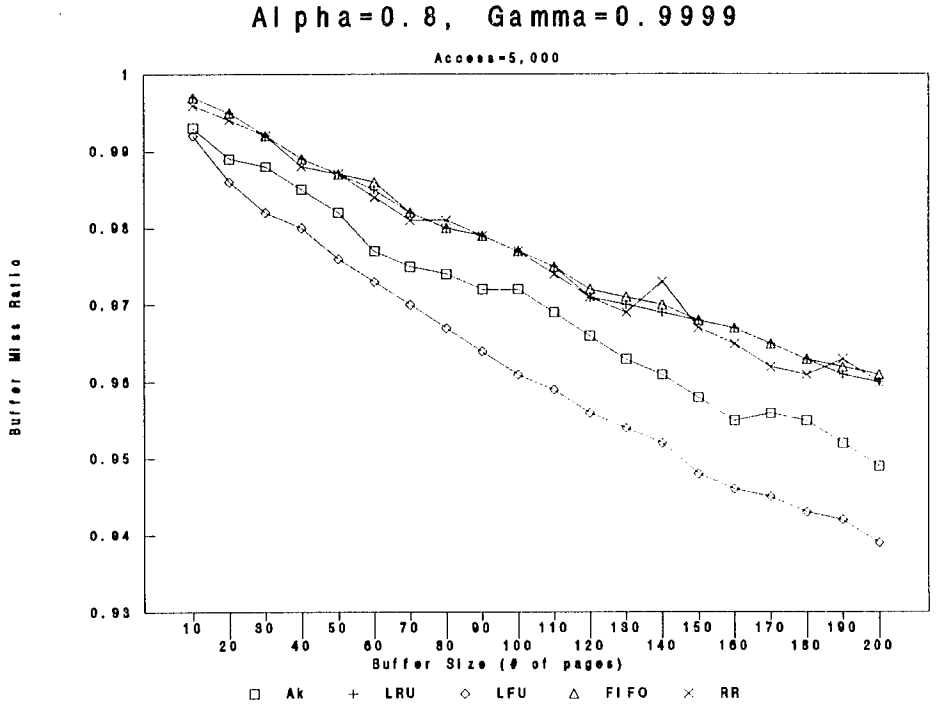

Figure 33: Relative Performance under α = 0.8, γ = 0.9999

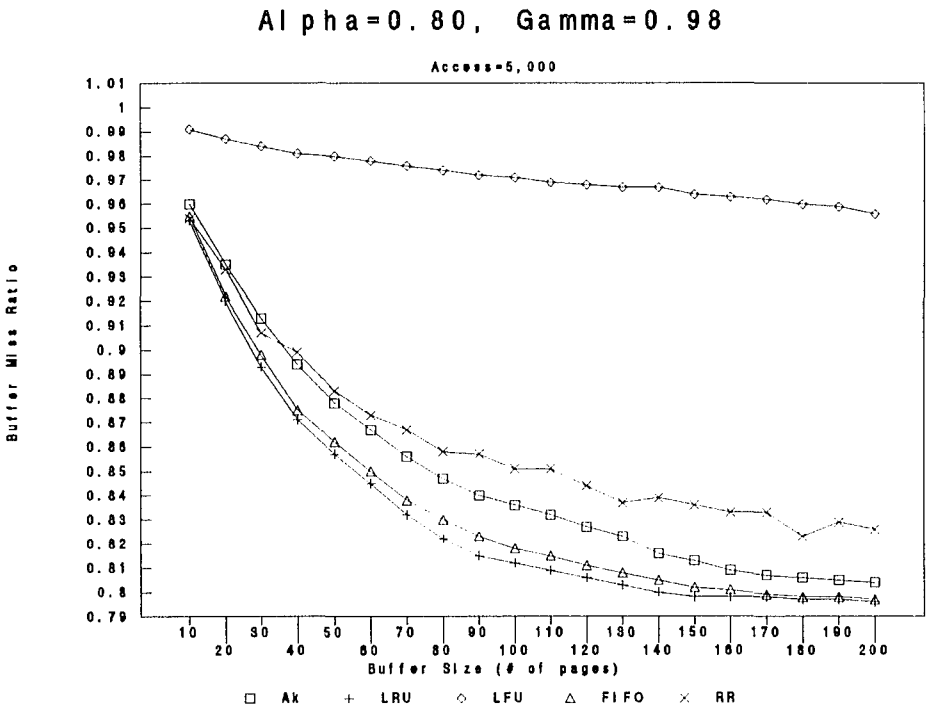

Figure 34: Relative Performance under α = 0.8, γ = 0.98

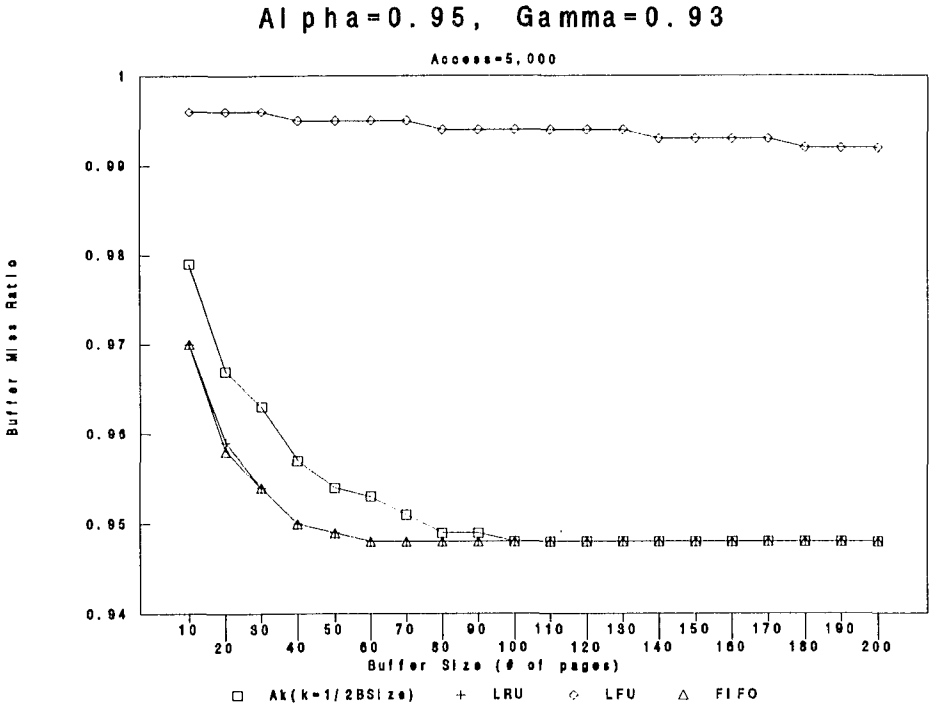

Figure 35: Relative Performance under α = 0.95, γ = 0.93

the LFU and A_K algorithms, but also the originally unreasonable LFU algorithm becomes the optimal algorithm. However, Figure 32 shows that a drop in the value of γ, even though slight magnitude, could completely change the relative performance of algorithms A_K, LRU and LFU. For example, when γ is set to be 0.99 instead of 0.9999, the A_K algorithm with K equal to half the buffer size outperforms all others, although the performance difference between A_K and the LRU is relatively small, or even nonexist, especially if the buffer size is reasonably large. In other words, the value of γ has a significant influence on the selection of optimal replacement algorithm.

When α is increased significantly (from 0.23 to 0.8, for instance) and γ is set to be almost 1 (0.9999 for instance), as in the case of Figure 33, the LFU algorithm obviously outperforms all other algorithms and is the optimal algorithm under this specific condition. This condition corresponds to the cases that when the degree of autocorrelation among consecutive accesses increases, and the probability of accessing a new record becomes very high during each access, the LFU algorithm will be the best choice among all other algorithms. This in fact could also be explained intuitively from the definition of the LFU algorithm.

Figure 34 and Figure 35 show that if the value of γ is changed from 0.9999 to 0.98, that is, decrease the degree of autocorrelation among consecutive accesses while the probability of accessing a new record is still high, the FIFO algorithm could be close to optimal, or at least comparable to that of the LRU algorithm. Our computational experiment result has shown that the range of parameter values is very narrow in order for the FIFO to be near optimal. On the other hand, the cases which the LRU algorithm is the optimal is more pervasive, as it was also shown in the last chapter.

The above observations of selecting a practical optimal algorithm given accessing model's parameter values also have practical meaning. Since the values of parameters α and γ all have their counterparts in real world applications, this simply means that an optimal replacement algorithm depends on the characteristics of database reference strings, i.e, the probability of accessing a new record, and also the degree of autocorrelation among the consecutive accesses. Our findings above is also generally consistent with our expectation when we think about the reference strings in terms of probability of new entry and degree of autocorrelation instead of pure numerical values.

5.3 Implementing Adaptive Replacement

Based on the previous study of the performance of various database buffer replacement algorithms, the selection of an optimal algorithm can be summarized in the table below:

Optimal Replacement Algorithm		γ	
		< 0.999	≥ 0.999
α	< 0.4	LRU/A_K	LFU
	$0.4 \leq \alpha < 0.9$	LRU	LFU
	≥ 0.9	FIFO/LRU	LFU

Notice that the values of parameters α and γ specified here are from the computational experimentation results obtained so far, with buffer size ranges from 10 to 200 pages and the number of accesses assumed to be 5,000. The accuracy of these ranges may be further improved after more extensive computational experimentation are carried out. In fact, it is much easier to find out the optimal replacement algorithm if we are given the values of parameters α and γ. Therefore, this table should serve as a general guideline in selecting an optimal replacement algorithm and special care need to be given if the parameter values are marginal.

The dual presentation of replacement algorithms in the two grids in the table means that both algorithms could give optimal or near optimal performance, in other words, the two algorithms have nearly the same performance. For example,

when $\alpha<0.4$ and $\gamma<0.999$, A_K with K equals to half the buffer size should also be one of the candidates for the optimal replacement algorithm, together with the LRU algorithm. Thus, consideration should be given to both algorithms under such situations.

The final selection of the optimal algorithm may also depend on certain other factors or specific requirements of an application, such as the complexity of implementing the specific algorithm. For example, we know that the Count heuristics in self-organizing linear search has certain extra space requirement to implement, this same requirement also presents in the implementation of the LFU buffer replacement algorithm. Therefore, A_K replacement algorithm could be the choice over the LFU algorithm when γ is in the range where these two algorithms have similar performance while α is in the smaller range (less than 0.4 for example), given that space is limited for the application.

As we know, Simon's modeling process of information accessing requires only a few parameters, but it allows access frequencies to be represented with much greater accuracy and could practically be applied to a lot of various application environments. Given the table about the selection of an optimal replacement algorithms above, the implementation of the self-adaptive buffer replacement scheme will then base on the observations of characteristics

in database reference strings, or in other words, the estimation of parameter α and γ of the accessing sequence.

One procedure of estimating the values of model parameters from the available empirical data graphically has been reported recently (Chong 1994). In general, the access frequencies generated by an operational system can be easily obtained by direct measurement and then used to estimate the model parameters. Researchers (Casas and Sevcik 1989) also indicate that for a system in the design stage, relative frequencies of accesses can be estimated based on the Bradford or Zipfian distributions which in turn be used to estimate the parameter values under Simon's model. In addition, the parameters of a system in design stage can also be estimated by the observations of similar operational systems.

Since the procedure of self-adaptive replacement should be automated and implemented during the processing of database applications. A method of estimating parameters α and γ is feasible only if the database system can estimate the parameter values dynamically during the system's normal operation.

To facilitate the automatic process of estimating the parameters of database accessing sequence, we keep the previous assumptions of the study and assume that the parameters of accessing model are estimated after every

5,000 accesses, in other words, the estimating procedure is executed in a batch size of 5,000 accesses. Obviously, the parameter α can be easily estimated from the ratio of the number of new records accessed within the current batch to that of the batch size.

The estimation of γ will then base on the value of α obtained as well as the observation of records with the highest frequency of access. This is due to the fact that parameter γ influences the degree of autocorrelation among consecutive accesses, or in other word, the concentration of accesses to database records. In fact, the frequencies of accesses obtained from the system in operation are compared with the sample simulation data obtained in advance from Simon's model of information accessing. Examples of such data generated from simulation are given in the table below.

Highest Access Frequency (Avg. of Top 3)		γ		
		0.99	**0.999**	**0.9999**
α	**0.2**	251 (168.0)	933 (441.7)	1120 (517.3)
	0.4	48 (35.0)	128 (99.3)	179 (138.3)
	0.6	18 (14.7)	34 (30.7)	48 (45.3)
	0.8	9 (8.0)	12 (10.0)	16 (10.3)

Note that in most cases, the access frequencies with highest value vary significantly as γ changes. Similar phenomenon can also be observed from the average of the top three records with highest accessing frequencies. In

addition, since the way parameter γ affects the optimal replacement algorithm depends only on whether the value is in high or low range — greater or less than 0.999, the result of estimation using this method should be reasonably accurate for implementation.

5.4 Future Applications

The concept of the self-adaptive replacement scheme and its variations can be applied to many other fields, including the more complicated cases in archival database design. Recent reports (Graefe 1993; Silberschatz et al. 1991) indicate that the next-generation database applications will involve much more data than today's business databases. As a result, the capability of efficient archival storage and archiving aging data to support the efficient access and modification of very large amount of data will no longer be a matter of choice. In fact, the problem of archival storage management is to schedule the primary and secondary storage, or in more complicated cases, multilevel storage in information systems so as to reduce storage costs and improve the system performance. This problem can be treated as a generalized case in buffer management, where the general guideline is to keep the most frequently used data as accessible to the users as possible.

Among the many problems that need to be studied in designing and managing archival storage, the selection of data to be removed ("retired") to archives in secondary storage media is one of those which has significant impact on the performance of database applications. The general performance objective is to minimize the probability of accessing information in the archival database, which is essentially similar to minimizing the miss ratio in buffer replacement. Therefore, the findings of our research in database buffer replacement can be directly extended to archival storage management, which in turn, improve the performance of future larger database systems.

As it is also reported (Silberschatz et al. 1991), next-generation database applications will have little in common with today's business data processing databases. In addition to much more data involved, they will also require new capabilities including multimedia support and complex objects. For example, currently, most insurance firms have a substantial on-line database that records the policy coverage of the firm's customers. These databases will soon be enhanced with multimedia data such as photographs of properties damaged, digitized images of handwritten claim forms, audio transcripts of appraiser's evaluations, images of specially insured objects, and so on. Since image data is exceedingly large, such databases will become enormous.

Moreover, other features like video walk through of houses will further enlarge the size of this class of databases. Overall, these new database applications will necessitate rethinking the algorithms for almost all DBMS operations, and our self-adaptive scheme should be able to adjust to this type of applications much more efficiently than the traditional algorithms.

5.5 Summary

In this chapter, we proposed a self-adaptive database replacement scheme based on the study of the optimal replacement algorithms under Simon's model of information accessing. We first demonstrated through computational experimentation that the optimal selection of replacement algorithms is very sensitive to the parameter values in Simon's model, with parameter γ having a more significant impact on the practical optimal replacement algorithm. We then outlined the general consideration in estimating the model parameter values for practical database reference strings in order to apply our findings in selecting the optimal algorithm.

The key idea of our self-adaptive replacement scheme is that since the performance of any replacement algorithm depends on the characteristics of database reference strings, the selection of a corresponding optimal replacement algorithm must thus be based on the estimation

of the characteristics of the database accesses and adapt itself to the application's environment. In addition, the buffer replacement scheme we proposed is fairly simple in general and incurs little overhead in terms of implementation. The most important feature is that this replacement scheme does not rely on external hints about reference strings' characteristics and adapts in real time to changing patterns of access.

CHAPTER 6 CONCLUSION AND FUTURE RESEARCH

In this dissertation, we first applied Simon's model of information accessing to generate record accessing frequencies of linear lists as well as database systems. The approach we proposed relaxes the independent assumption in modelling database accessing (i.e., the IRM model) which was commonly used in previous studies and also allows certain dynamic behavior in accessing frequencies. We showed that this approach is more robust and preferable over the traditional "artificial data" approach. As a result, tighter bounds of the relative performance of self-organizing heuristics were obtained, which we believe is more realistic over the previous analytical results.

Through studying the self-organizing heuristics in linear search and their conceptual similarity with the traditional buffer replacement algorithms, we proposed a new class of database replacement algorithms which are conceptually similar to the Move-Ahead-K heuristics. We have shown that this newly proposed replacement algorithms have some unique characteristics compared with the traditional algorithms, which in turn, display certain performance advantages under some database reference strings.

Finally, we proposed a self-adaptive buffer replacement scheme that adjusts itself to the patterns of database references in selecting the optimal replacement algorithm. The optimization potential of our self-adaptive replacement scheme is based on the fact that since the scheme is self-adaptive, it can adjust itself to a specific DBMS and an application environment. The basic trade-off here is the conceptual simplicity of the traditional algorithms versus the potential improvement of database performance under our new scheme. Future applications of this efficient self-adaptive methodology were also proposed so that we could extend our findings to improve the performance of advanced information systems.

Our research work in this dissertation should be viewed as an initial attempt to optimize the database buffer replacement. While the desirability of a replacement scheme that satisfies the diversity of application environments has been shown, the evaluation of the replacement algorithms, the estimation of database access patterns, and the applicability of the self-adaptive replacement scheme should be more extensively explored. Specifically, our results should be extended to a wider range of replacement algorithms, for example, wider range of parameter K in A_K algorithms may be studied. Also the design of storage structure should be studied since the replacement algorithms

have a close relationship with buffer management in general, and particularly secondary storage management.

Because the eventual purpose of this study is to automate the process of self-adaptive buffer replacement, other problems of special interests for future research would be, for example, how can the knowledge of applications be made available for the predication of future database reference behavior? Some means have to be introduced to allow the DBMS to accept "advice" from the query interpretation or compilation process, for example, in order to make full use of the context information of high-level database languages. Only after this process of acquiring the knowledge of database reference behavior has been automated, will the self-adaptive replacement scheme be practically implemented on commercial database systems.

REFERENCES

Anderson, E.J., P. Nash, and R.R. Weber, "A Counter Example to a Conjecture on Optimal List Orderings," *Journal of Applied Probabilities*, Vol. 19, 1982, pp. 730-732.

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

Bellow, M.E., "Autoregressive Performance Analysis of Self-organizing Data Structures," *Computer Science and Statistics: Proceedings of the 19th Symposium on the Interface*, Philadephia, PA, USA, March 8-11, 1987, pp. 417-421.

Bentley, J. L. and McGeoch, C. C., "Amortized Analyses of Self-organizing Sequential Search Heuristics," *Communications of the ACM*, Vol. 24, No. 4, 1985, pp. 404-411.

Berg J., and D. Towsley, "Properties of the Miss Ration for a 2-Level Storage Model with LRU or FIFO Replacement Strategy and Independent References," *IEEE Transactions on Computers*, Vol. 42, No. 4, April 1993.

Bitner, J.R., "Heuristics that Dynamically Organize Data Structures," *SIAM Journal of Computing*, Vol. 8, No. 1, 1979, pp. 82-110.

Bitner, J.R., "Heuristics That Dynamically Alter Data Structure to Reduce Their Access Time," *Ph.D thesis*, University of Illinois, 1976.

Burville, P.J. and J.F.C. Kingman, "On a Model for Storage and Search," *Journal of Applied Probabilities*, Vol. 10, 1973, pp. 697-701.

Casas, I.R. and K.C. Sevcik, "A Buffer Management Model for Use in Predicting Overall Database System Performance," *Proceedings of the 5th International Conference on Data Engineering*, Los Angeles, California, February 1989, pp. 463-469.

Cheetham R. P., B. J. Oommen, and D. T.H. Ng, "Adaptive Structuring of Binary Search Trees Using Conditional

Rotations," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, August 1993.

Chen, Y.S., "Analysis of Lotka's law: the Simon-Yule approach," *Information Processing & Management*, Vol. 25, No. 5, 1989, pp. 527-544.

Chen, Y.S., "An exponential recurrence distribution in the Simon-Yule model of text," *Cybernetics and Systems: An International Journal*, Vol. 19, 1988, pp. 521-545.

Chen, Y.S., P.P. Chong, and Y. Tong, "Theoretical Foundation of the 80/20 Rule," *Scientometrics*, Vol. 28, No. 2, 1993, pp.183-203.

Chen, Y.S., P.P. Chong, and M.Y. Tong, "The Simon-Yule Approach to Bibliometric Modeling," *Information Processing & Management*, forthcoming in 1994.

Chen, Y.S. and Leimkuhler, F.F., "Booth's Law of Word Frequency," *Journal of the American Society for Information Science*, Vol. 41, No. 5, 1990, pp. 387-388.

Chen, Y.S., and F.F. Leimkuhler, "A Relationship Between Lotka's Law, Bradford's Law, and Zipf's Law," *Journal of the American Society for Information Science*, September 1986, pp. 307-314.

Ch'ng, H.T., B. Srinivasan, and B.C. Ooi, "Study of Selforganizing Heuristics For Skewed Access Patterns," *Information Processing Letters*, March 1989, pp. 237-244.

Ch'ng, H.T., and B. Srinivasan, "Permutation Algorithms and Data Structures For Exploiting Skewness in Data Access," *Proceedings of the Eleventh Annual International Computer Software & Applications Conference*, Tokyo, Japan, October 7-9, 1987, pp. 439-444.

Chong, P.P, "On Information Usage Modeling," Ph.D. dissertation, Louisiana State University, 1994.

Chung, F.R.K., D.J. Hajela, and P.D. Seymour, "Self-organizing Sequential Search and Hilbert's Inequalities," *Journal of Computer and System Science*, Vol. 36, 1988, pp. 148-157.

Coffman, E.G. and P.J. Denning, *Operating Systems Theory*, Prentice-Hall, Englewood, NJ, 1973.

Courcoubetis, C., and R.R. Weber, "The Move-to-Front Rule for Multiple Lists," *Probability in the Engineering and Informational Sciences*, Vol.4, No. 1, 1990, pp. 19-27.

Dan, A., and D. Towsley, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes,", *Proceedings of the 1990 ACM SIGMETRICS Conference*, Vol. 18, No. 1, 1990, pp. 143-149.

Effelsberg, W. and T. Haerder, "Principles of Database Buffer Management," *ACM Transactions on Database Systems*, Vol. 9, No. 4, December 1984, pp. 560-595.

Fenwick, P.M., "A New Technique for Self-Organising Linear Searches," *The Computer Journal*, Vol. 34, No. 5, 1991, pp. 450-454.

Gonnet, G., J.I. Munro, and H. Suwanda, "Exegesis of Self-Organizing Sequential Search Heuristics," *SIAM Journal Computing* Vol. 10, 1981, pp. 613-637.

Graefe, G., "Options in Physical Database Design," *SIGMOD Record*, Vol.22, No.3, September 1993.

Heising, W.P., "Note on Random Addressing Techniques," IBM Sysholdings Journal, Vol. 2, No. 2, June 1953, pp. 112-116.

Hendrick, W.J., "An Account of Self-organizing Systems," *SIAM Journal Comput*. Vol. 5, 1976, pp. 715-723.

Hester, J.H., and Hirschberg, D.S., "Self-organizing Linear Search," *Computing Surveys*, Vol. 17, No. 3, 1985, pp. 295-311.

Hossain, A., A.R. Marudarajan, and M. A. Manzoul, "Fuzzy Replacement Algorithm for Cache Memory," *Cybernetics and Systems: An International Journal*, Vol. 22, 1991, pp. 733-746.

Hui, L.C.K, and C. Martel, "Unsuccessful Search in Self-Adjusting Data Structures," *Journal of Algorithms*, Vol.15, 1993, pp. 447-481.

Ijiri, Y., and H.A. Simon, *Skew Distributions and the Sizes of Business Firms*, North-Holland, 1977.

Kan, Y.C., and S.M. Ross, "Optimal List Order Under Partial Memory Constraints," *Journal Applied Probability*, Vol. 17, 1980, pp. 1004-1015.

Kaplan, J., "Buffer Management Policies in a Database System," M.S. Thesis, Univ. of California, Berkeley, 1980.

Kearns, J.P., and S. DeFazio, "Diversity in Database Reference Behavior," *Performance Evaluation Review*, Vol. 17, No. 1, May 1989, pp. 11-19.

Kim, Jin-Soo, "Usage-dependent Information Systems Design," Ph.D. dissertation, Louisiana State University, 1990.

Knuth, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addision-Wesley, Reading. Mass. 1973.

Lang, T. and I.B. Fernandez, "Database Buffer Paging in Virtual Storage Systems," *ACM Transactions on Database Systems*, Vol. 2, No. 4, December 1977, pp. 339-351.

Lau, E. J., *Performance Improvement of Virtual Memory Systems*, UMI Research Press, 1982.

Leimkuhler, F.F., "On bibliometric modeling," *Informetrics*, 1988, pp. 97-104.

Makinen, E., "On Linear Search Heuristics," *Information Processing Letters*, September 1988, pp. 35-36.

Mandelbrot, B., "An information theory of statistical structure of language," *Proceedings of the Symposium on Applications of Communications Theory*, (London, September 1952), London: Butterworths, 1953, pp. 486-500.

Martel, C., "Self-adjusting Multi-way Search Trees," *Information Processing Letters*, Vol. 38, 1991, pp. 135-141.

Mattson, R.L., J. Gecsei, D.R. Slutz, and I.L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM System Journal*, Vol. 9, No. 2, 1970, pp. 78-117.

McCabe, J., "On Serial File with Relocatable Records," *Operations Research*, Vol. 12, 1965, pp.609-618.

Naor, D., C.U. Martel, and N.S. Matloff, "Performance of Priority Queue Structures in a Virtual Memory Environment," *The Computer Journal*, Vol. 34, No. 5, 1991.

Neuts, M.F., "Computer Experimentation in Applied Probability," working paper 86-030, Systems and Industrial Engineering Department, University of Arizona, 1986.

Ng, D.T.H., and B.J. Oommen, "A Short Note on Doubly-Linked List Reorganizing Heuristics," *The Computer Journal*, Vol. 35, No. 5, 1992, pp. 533-535.

Nicola, V.F., A. Dan, and D.M. Dias, "Analysis of the Generalized Clock Buffer Replacement Scheme for Database Transaction Processing," *Performance Evaluation Review*, Vol. 20, No. 1, June 1992, pp. 35-46.

O'Neil, E.J., P.E. O'Neil, and G. Weikum, "The LRU-K Page Replacement Algorithm For Database Disk Buffering," *Proceedings of the 1993 ACM SIGMOD Conference*, May 1993, pp. 297-306.

Oommen, B.J., and E.R. Hansen, "List organizing strategies using stochastic move-to-front and stochastic move-to-rear operations." Technical Report SCS-TR-76, School of Computer Science, Carleton University , Ottawa, Canada, 1985.

Palvia, P., "Approximating Block Accesses in Random Files: The Case of Blocking Factors Lower Than One," *Information Systems*, Vol. 16, No. 3, 1991, pp. 357-361.

Rivest, R., "On Self-Organizing Sequential Search Heuristics", *Communications of the ACM*, Vol. 19, No. 2, 1976, pp. 63-67.

Sanders, R., "The Pareto Principles: Its Use and Abuse," The Journal of Services Marketing," Vol. 1, No. 2, Fall 1987, pp. 37-40.

Sacco, G.M. and M.S. Schkolnick, "Buffer Management in Relational Databases Systems," *ACM Transactions on Database Systems*, Vol. 11, No. 4, December 1986, pp. 473-498.

Silberschatz, A., M. Stonebraker, and J. Ullman, "Database Systems: Achievements and Opportunities," *Communications of the ACM*, Vol. 34, No. 10, October 1991.

Simon, H.A., "On judging the plausibility of theories, in B. van Rootselaar and J. F. Staal (eds.)," *Logic,*

Methodology and Philosophy of Sciences, Vol. III, Amsterdam: North-Holland, 1968.

Simon, H.A., "On a Class of Skew Distribution Function," *Biometrika*, Vol. 42, 1955, pp. 425-440.

Simon, H.A., and T.A. Van Wormer, "Some Monte Carlo Estimates of the Yule Distribution," *Behavior Science*, Vol. 1, No. 8, 1963, pp. 203-210.

Sleator, D.D., and Tarjan, R.E., "Self Adjusting Binary Trees," *Journal of the ACM*, Vol. 32, No. 3, 1985, pp. 652-686.

Sleator, D.D., and Tarjan, R.E., "Amortized Efficiency of List Update and Paging Rules," *Communication of the ACM*, Vol. 28, No. 2, February 1985, pp. 202-208.

Smith, A.J., "Sequentiality and Prefetching in Database Systems," *ACM Transactions on Database Systems*, Vol. 3, No. 3, September 1978, pp. 223-247.

Stonebraker, M., "Operating System Support for Database Management," *Communications of the ACM*, Vol. 24, No. 7, July 1981, pp. 412-418.

Tenenbaum, A., "Simulations of Dynamic Sequential Search Algorithms," *Communications of the ACM*, Vol. 21, No. 9, 1978, pp. 790-791.

Tenenbaum, A.M. and R.M. Nemes, "Two Spectra of Self-Organizing Sequential Search Algorithms," SIAM Journal of Computing, Vol. 11, No. 3, August 1982, pp. 557-566.

Valiveti, R.S., and Oommen B.J., "Self-Organizing Doubly-Linked Lists," *Journal of Algorithms*, Vol.14, 1993, pp. 88-114.

Zipf, G.K., *Human Behavior and the Principal of Least Effort*, Cambridge, MA, Addison-Wesley, 1949.

VITA

Yueguo Tong was born on May 21, 1963 in Shanghai, P. R. China. He received his B.S. and M.S. in Computer Science from Fudan University, Shanghai in 1985 and 1989 respectively. Upon finishing his course works for the M.S. degree, he was appointed Software Engineer in Shanghai Computer Software Laboratory and later Marketing Manager of Shanghai Broad Technology Cooperation. He came to the United States in April 1990 and was admitted to the Ph.D. program in the Department of Quantitative Business Analysis at Louisiana State University in Fall 1990.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Yueguo Tong

Major Field: Business Administration

Title of Dissertation: A Self-Adaptive Database Buffer Replacement Scheme

Approved:

Ye-Sho Chen
Major Professor and Chairman

Daniel Fogel
Dean of the Graduate School

EXAMINING COMMITTEE:

Peter Kelle

Ishwar Murthy

Sumit Sarkar

Kwei Tang

Lonnie R. Vandeveer

Date of Examination:

May 17, 1994