Information Systems Vol. 17, No. 6, pp. 491–509, 1992
Printed in Great Britain.

0306-4379/92 $5.00 + 0.00

FILE ORGANIZATIONS WITH SHARED OVERFLOW BLOCKS FOR VARIABLE LENGTH OBJECTS†

YANNIS MANOLOPOULOS[1] and STAVROS CHRISTODOULAKIS[2]

[1]Department of Electrical Engineering, Aristotle University of Thessaloniki, 54006 Thessaloniki, Greece

[2]Department of Electronic and Computer Engineering, Technical University of Crete, 73133 Chania, Greece

(Received 9 September 1991; in revised form 31 August 1992)

Abstract—Traditional file organizations for records may also be appropriate for the storage and retrieval of objects. Since objects frequently involve diverse data types (such as text, compressed images, graphics, etc.) as well as composite structures, they may have a largely variable length. In this paper, we assume that in the case of composite objects their components are clustered together and that object file organizations have overflows. The blocks of the main file are grouped so that they share a common number of overflow blocks. For this class of file organizations we present and analyze the performance of three different overflow searching algorithms. We show that the third algorithm gives very significant performance advantages under certain circumstances.

Key words: File structures, variable length objects, chaining, overflow area, successful and unsuccessful search, algorithms, performance analysis

1. INTRODUCTION

Various file organizations utilize overflow areas such as the recent developments of hashing [1] (several variations of linear and dynamic hashing, bounded index exponential hashing, hashing with interpolation-based index maintenance, etc.) as well as the traditional indexed sequential files [2, 3]. At loading time data can almost uniformly be distributed in blocks but insertions and deletions hit blocks with different probabilities. Therefore, it is almost certain that after some time a number of main file blocks will demand excess overflow space while others will not. Overflowing means that sooner or later the performance deteriorates due to additional required accesses. Reorganization will take place either locally and dynamically or globally and periodically as in the cases of hashed and indexed sequential files respectively.

Traditional file organizations for records may also be appropriate for the storage and retrieval of objects occurring in object oriented languages, nested relations and extensible database systems which use fixed size pages. Objects in our model have a key according to which they are filed. The other object fields and their encoding are not important (and may not be known) to the file organization. However, since objects frequently involve diverse data types (such as text, compressed images, graphics, etc.) as well as composite structures, the common practice is that they have a largely variable length.

A basic assumption of this work is that the objects may have a highly variable length. Although variable length records are very frequent in conventional database environments due to variable lenght fields, missing attribute values, multiple values of an attribute and compression, little has been reported in the literature about their effect on the file performance. We note the works of Hakola and Heiskanen [4], Hubbard [5], Manolopoulos and Faloutsos [6], Teorey and Fry [7] and Wiederhold [8] on estimating the wasted space at the end of blocks due to variable length records. New database trends, towards multimedia and object oriented databases, emphasize the need for studies of the effect of length variability on the performance of file organizations. The reason is that the new data types (text, graphics, images) have a very large length variance. A recent relevant work is [9], which examines the space overhead due to the variability of object lengths involved in some physical storage models for use in object-oriented databases, such as IRIS [10], ORION

†To avoid any delay in publication of this issue, this paper has been published without the authors' corrections.

[11], TAXIS [12] etc. Here, we assume that in the case of composite objects their components are clustered together.

In this paper we concentrate our attention on object file organizations which have overflows. Recently, some analysis and experimentation on various overflow strategies for handling variable length records were reported [2]. More specifically large overflow blocks were assumed and analysis on the search performance was carried out. In the present report we assume that the blocks of the main file are grouped so that they share a number of common overflow blocks. For this class of file organizations we present and analyze the performance of three different overflow searching algorithms. Two of them are new and improve the algorithm presented in [2]. In addition, we show that the third algorithm gives very significant performance advantages when:

- the main blocks are grouped in small numbers,
- there is large variability of object size and
- the object sizes are relatively small when compared with the block size.

These algorithms may be applied in almost any file organization that utilizes an overflow area.

The rest of the paper is organized as follows. In Section 2 the basic mathematical model for the specific file organization is developed. The probability distribution of the number of objects per area (main block and attached overflow objects) for two structures (hashed and indexed sequential) is given as a function of file expansion with time. In the same section the probability distribution of objects in the main block and in the overflow area is given. The material of the section has been reported previously, mainly in [2], but it is included here very briefly because it is essential for the analysis to follow. For more details on the derivations of the analysis the interested reader is directed to the reference. In Section 3 the three algorithms are presented and mathematically analyzed. Formulae for the successful and unsuccessful search are derived as a function of the file expansion with time. In Section 4 figures illustrating performance comparisons of the algorithms are discussed. Finally, some conclusive remarks follow.

2. MATHEMATICAL BASIS

File objects are divided into classes $C_1, C_2, \ldots, C_L$. We consider a finite number of classes, L, for simplifying the analysis. Any arbitrary length object is mapped into the class which has a length nearest to its length. Class C_i contains objects with length l_i and known arrival probability p_i, where $i = 1, 2, \ldots, L$. Object lengths are assumed to be independent of the key values. Let x be the number of objects that exist in an area.

Lemma 1. The probability that n_1 of these x objects have been selected from class C_1, n_2 from class $C_2, \ldots, n_L$ from class C_L obeys a multinomial distribution:

$$q(n_1, \ldots, n_L) = \frac{x!}{n_1! \ldots n_L!} p_1^{n_1} \ldots p_L^{n_L}$$

The probability distribution of the number of objects in an area (main block and overflow objects) depends on the file structure and is different for the hashed or the indexed sequential files. For the sequel, suppose that a file which consists of NB main file blocks was initially loaded with M objects. After some time N additional objects are stored in the file, thus making a total of $X = M + N$ objects. Table 1 gives the definitions of all the symbols used throughout this paper.

An approximate probability mass distribution assigned to a block and the probability distribution of the number of records per block of an indexed sequential file at some point in time after initial loading was first derived by Larson [3]. An alternate exact analysis was reported by Batory [13]. However, in [2] a new asymptotic probability distribution was derived and it was shown to be a negative binomial distribution.

Lemma 2 [2]. The probability that at some point in time x objects out of the X ones are stored in a specific main block (out of the NB main blocks of an indexed sequential file) and its associated overflow ones is:

$$P_m(x, a) = \binom{x-1}{n} \frac{a^n}{(1+a)^x}$$

Table 1. Symbol definitions

Symbol	Definition
m	Number of objects initially loaded in a block
n	Number of objects inserted in a block
M	Total number of objects initially loaded in the file
N	Total number of objects inserted into the file
a	File expansion vector
BS, KS, PS	Block, Key and Pointer sizes (in bytes)
L	Number of classes
p_i	Probability distribution of the number of objects belonging to class i($1 \leqslant i \leqslant L$)
l_i	Length of object belonging to class i ($1 \leqslant i \leqslant L$)
$l_{ov}(a)$	Average overflow object length (function of a)
max	Maximum number of objects in a block
NB	Number of main blocks after loading
MB	Number of main blocks sharing the same overflow blocks
U	Average used space in an overflow block (in bytes)
$P(x, a)$	Probability that a block contains x objects (function of a)
$X(a)$	Average number of objects in a block (function of a)
$q(n_1, \ldots, n_L)$	Probability that n_i objects have length i ($1 \leqslant i \leqslant L$)
$Q(b, x)$	Probability that exactly b out of x objects are stored in the main block
$P_i^{int}(a)$	Probability distribution that an object class i ($1 \leqslant i \leqslant L$) is intercepted by a block boundary (function of a)
$P_i^{ov}(a)$	Probability distribution of overflow object class i ($1 \leqslant i \leqslant L$)
$N_{ov}(a)$	Average number of overflow objects per MB main blocks (function of a)
$MB_{ov}(a)$	Average number of shared overflow blocks per MB main blocks (function of a)
b	Average number of objects per overflow block
$B(k)$	Average number of overflow blocks containing k objects of an area
$L_s(k)$	Average number of accesses for a successful search in a chain of k objects
$L_u(k)$	Average number of accesses for an unsuccessful search in a chain of k objects
$S(a)$	Average successful search cost (function of a)
$U(a)$	Average unsuccessful search cost (function of a)

where m out of M is the number of objects initially stored in this block, n out of N is the number of additional objects stored in the block after certain time and a, the file expansion factor, is equal to N/M.

If a uniform hashing function is assumed to be employed in a hash based file organization, then the probability distribution of the number of objects per block is the binomial one. In this case the following lemma holds.

Lemma 3. The probability that at some point in time x objects out of the X ones are stored in a specific main block (out of the NB main blocks of hashed file) and its associated overflow blocks is:

$$P(x, a) = \binom{(a+1)M}{x} \left(\frac{1}{NB}\right)^x \left(1 - \frac{1}{NB}\right)^{(a+1)M - x}$$

Let b be the number of objects which are stored in the main block. This number is a random variable and depends on the distribution of object lengths. Let $Q(b, x)$ be the probability that exactly b out of the x objects that exist in an area are stored within the main block. The variable $Q(b, x)$ is given by the expression of the following lemma.

Lemma 4 [2]. The probability that exactly b out of the x objects that exist in an area are stored within the main block is:

$$Q(b, x) = \sum_{\Sigma_{i=1}^{L} n_i l_i \leqslant BS - PS} q(n_1, \ldots, n_L) \sum_{l_i + \Sigma_{j=1}^{L} n_j l_j > BS - PS} p_i$$

where $\Sigma_{i=1}^{L} n_i = b \leqslant x$, BS is the number in bytes available per main or overflow blocks, PS is the size of a pointer in bytes and KS is the size of the key in bytes (fixed).

At load time empty space is distributed in main blocks to accomodate future insertions but sooner or later the main block becomes full and overflowing is unavoidable. Objects are not allowed to span over two different blocks; therefore some space is left unused. From each main block only one chain may emanate. Objects in a main block and its overflow chain are sorted according to key values. This technique is a rule for indexed sequential files but it may be applied in hashed files too, as in [14, 15], with better search performance measures at the cost of more expensive insertions. The highest keys are placed in the overflow chain. When an object is inserted in the main block, the last object of the block may not fit any more in it and therefore it will have to be

transferred to the overflow chain. In a similar manner, if an object of the main block is deleted, then if the first object of the overflow chain fits in the main block then it has to be stored in it.

The overflow block size has been considered to be equal to the main block size. This implementation agrees with the current systems which use preformatted disks (disks with fixed block size). Overflow objects are connected in a chain with pointers and several objects from different areas or the same area may exist in the same overflow block. In other words overflow blocks are shared by a varying number of main blocks. This number is a parameter of the file system and in the following will be depicted by MB. In [3] it was accepted that overflow block capacity is one record only. A similar scheme to the one proposed here with shared overflow blocks is employed by Yuen and Du [16] in a study of variations on linear hashing, designed for partial match retrieval. In Fig. 1 a sample of this overflow file scheme is illustrated.

It is asumed, also, that there is a DB-cache capability and therefore an overflow block remains in main memory, so that it does not have to be retrieved more than once if more than one object of the overflow chain exist in the block. This capability results in reduced search costs. According to our scheme, since the overflow blocks are large, variable length objects may be packed better within a block resulting in better space utilization and reduced search cost.

Lemma 5 [2]. The probability that an object intercepted by a block boundary is of type C_i is:

$$P_i^{int}(a) = \frac{P_i(a)}{\sum_{j=1}^{L} P_j(a)}$$

where:

$$P_i(a) = \sum_{x=2}^{max} P(x, a) \sum_{\Sigma_{j=1}^{L} n_j l_j \leq BS - PS \leq l_i + \Sigma_{j=1}^{L} n_j l_j} q(n_1, \ldots, n_L) p_i$$

$\Sigma_{h=1}^{L} n_j < x$, whereas the maximum number of objects which may reside in the main block is denoted by max.

Proof. The probability distribution of the lengths of objects in the overflow blocks is not in general the same as the probability distribution of the lengths of objects in the main block. The reason is that longer objects have higher probability to be intercepted by the main block boundary than shorter objects have. Thus longer objects are more likely to be found in the overflow chain. Let $P_i(a)$ be the total probability of any arrangement of objects in the main block so that an object of type C_i is intercepted by a block boundary. Then the relation of the theorem concerning $P_i(a)$ is easily explained. It is noted that max denotes the maximum number of objects which may reside in the main block and, evidently, it is equal to BS/l_{min}. Observe that $P_i(a)$ is a function of the file expansion factor since for small values of the file expansion factor only long objects may intercept the block boundary. This formula is derived in a manner similar to that of Lemma 4 taking into account that the object with the next in order key value may not fit within the block and therefore it may have to move in the overflow area. After normalization the relation of the lemma is derived. □

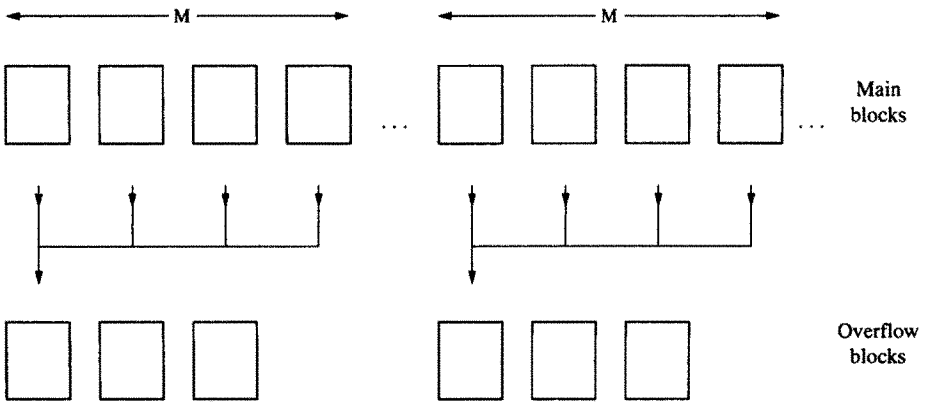

Fig. 1. Shared overflow blocks per $MB = 4$ main blocks.

Lemma 6. The probability distribution of the number of overflow objects is given by:

$$P_i^{ov}(a) = (P_i^{int}(a) - P_i) \frac{\sum_{x=2}^{\infty} P(x,a) \sum_{b<x} Q(b,x)}{\sum_{x=2}^{\infty} P(x,a) \sum_{b<x} (x-b)Q(b,x)} P_i$$

where $1 \leqslant i \leqslant L$.

Proof. To find the length distribution in the overflow file we have to consider both the intercepted objects as well as the remaining objects in the overflow chain (which follow the probability distribution P_i, where $i = 1, \ldots, L$). The proportion of the intercepted objects to the total number of objects in overflow blocks is:

$$\frac{MB \sum_{x=2}^{\infty} P(x,a) \sum_{b<x} Q(b,x)}{MB \sum_{x=2}^{\infty} P(x,a) \sum_{b<x} (x-b)Q(b,x)}.$$

Therefore the relation of the lemma holds. □

Lemma 7. The expected number of objects which overflow from MB main blocks can be estimated.

$$N_{ov}(a) = MB \sum_{x=2}^{\infty} P(x,a) \sum_{k=1}^{x-1} kQ(x-k,x)$$

Lemma 8. The expected number of shared overflow file blocks per MB main blocks is given by:

$$MB_{ov}(a) = \frac{N_{ov}(a) l_{ov}(a)}{U}$$

where $l_{ov}(a)$ is the average overflow length and U is the average number of bytes used within an overflow block.

Proof. The number of shared overflow blocks per MB main blocks depends on the algorithm which handles insertions and deletions in the file. Assuming that a very sophisticated algorithm exists which optimizes the space use in the overflow file by utilizing some packing algorithm (e.g. first fit, best fit) and that deletions are not very frequent, then as a rough approximation it may be accepted that the expected unused space is half of the mean object length [6] (with the exception of the last block). Under this assumption the expected number of shared overflow file blocks per MB main blocks is given by: $MB_{ov}(a) = N_{ov}(a) l_{ov}(a)/U$, where: $l_{ov}(a) = \Sigma_{i=1}^{L} P_i^{ov}(a) l_i$, is the average overflow object length and U is the average number of bytes used within an overflow block. Under the above assumptions U can be approximated by: $U = BS - l_{av}/2$, where l_{av} is the average object length. Evidently the following relation holds: $l_{av} = \Sigma_{i=1}^{L} p_i l_i$. □

It is however very difficult to calculate the probability distribution of the number of objects within a block. The reason is that the object occurence probabilities are not independent any more due to the packing algorithm. When the packing algorithm is not as good or when high deletion rates make the approximations unrealistic then the average space left unused per block and the probability distribution of the number of objects within the overflow file blocks can be estimated by using a simulation of the packing algorithm. Another possibility is that no packing algorithm is used, but objects are always placed at the end of the overflow file, and there are no deletions of objects from the file. A simplication of the problem that has been used in the past is to assume that a constant number of objects exists in each overflow block. It has been shown that this simplification may result in pessimistic performance estimates [17]. Therefore, the following lemmas hold.

Lemma 9. The expected number of objects per overflow block is:

$$b = \frac{N_{ov}(a)}{MB_{ov}(a)} = \frac{U}{l_{ov}(a)}$$

Lemma 10. The expected number of overflow blocks which contain k objects of an area is [18]:

$$B(k) = MB_{ov}(a) \quad 1 - \frac{\binom{N_{ov}(a) - b}{k}}{\binom{N_{ov}(a)}{k}}$$

Cost equations for the three search algorithms will be derived based on the previous formula.

3. OVERFLOW SEARCH ALGORITHMS AND THEIR ANALYSIS

Before proceeding to the analysis of the algorithms, we describe them, illustrate their differences and show the performance improvement by an example.

Algorithm 1

In the first algorithm we simply follow the pointers of the overflow chain until two cases may happen. Either the object with the specific key value is found (successful search) or it is evident that the object does not exist in the file (unsuccessful search), because objects with key values greater than the desired one are retrieved. Accessed overflow blocks are assumed to remain main memory resident due to a DB-cache capability so that a block that contains more than one objects of a specific overflow chain will be retrieved only once from secondary storage.

Algorithm 2

The above search algorithm is somewhat naive but when modified slightly it can result in better performance. In this modified algorithm, at each block access all the objects of the block are examined. If the desired object exists in this block the search terminates. Otherwise, the chain pointer from the object, for which the access was paid, to a successive block is followed. This simple modification of the algorithm improves the performance of the successful search but it does not affect the cost of the unsuccessful search.

Algorithm 3

An even more efficient algorithm in terms of expected block accesses can be thought for answering successful as well as unsuccessful queries. As in the second algorithm, the keys of all objects of a retrieved block are examined but in addition it is examined whether there is any object with a key value between the desired one and the value of the object for which the block access was made. For example, if a key value of an object j between the values of the ith and the rth object is found, when the block which contains the ith object is retrieved, the search may continue by following the chain emanating from this object.

Example. Assume that a chain of 6 sorted objects is stored in 4 overflow blocks as illustrated in Fig. 2. Suppose, also, that the object with key value 60 is searched. The first algorithm will fetch the blocks A, B, C and D which makes in total 4 block accesses. The second one will fetch 3 blocks, namely the A, B and C ones. This is due to the fact that, when block C is retrieved, all the objects of the block are examined and therefore the desired object with key 60 is found. The third algorithm will fetch only 2 blocks, A and C. This happens because, when block A is fetched, the pointer from the highest of the two keys (30) is followed and therefore the block C is retrieved. The search of block C finds not only the follow-up key (40) but also the desired key (60). Thus, in this example there is a performance improvement by using the more sophisticated algorithms.

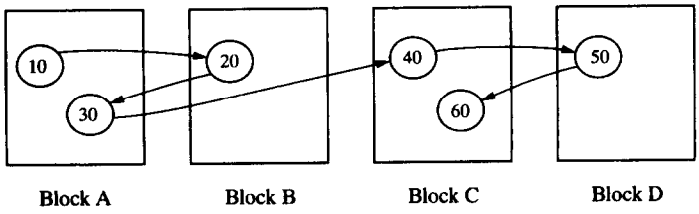

Fig. 2. Overflow chain of 6 objects residing in 4 blocks.

A second observation concerning the performance of the second and the third algorithm is that the search cost of the successive objects of a chain is not necessarily an increasing function of the chain length as opposed to the performance of the first algorithm. Later it will be shown that in the expected case the search cost is an increasing function. The sequence of search costs in block accesses for the successive objects of the chain of the layout of the Fig. 2 is:

1, 2, 2, 3, 4, 4 according to the first algorithm,
1, 2, 1, 3, 4, 3 according to the second and
1, 2, 1, 2, 3, 2 according to the third one.

A third observation has to do with the relation between successful and unsuccessful search. The 6 objects of the chain form 7 subintervals. Therefore, suppose we are going to search for 7 non-existing objects, each one lying in a different subinterval. For convenience we name these objects 5, 15, 25, 35, 45, 55 and 65 respectively. The sequence of search costs in block accesses for the list of these non existing objects is:

1, 2, 2, 3, 4, 4, 4 according to the first algorithm,
1, 2, 2, 3, 4, 4, 4 according to the second and
1, 2, 2, 2, 3, 3, 2 according to the third one.

We note that, as expected, the unsuccessful search cost, by using the first or the second algorithm, are identical. In other words, the search cost by using the second algorithm is a non decreasing function as we search for the successive objects of a chain. We note, also, for the third algorithm that the search cost of a non existing object is not less than the cost of the successful search of the preceeding existing object in the chain.

Next we derive estimates of the expected cost for the successful and unsuccessful search when the three search algorithms are used for finding a qualifying object.

3.1. First algorithm

The analysis of this case is straightforward.

Theorem 1. The expected number of additional block accesses for the successful search of any object out of the k ones of an overflow chain is given by [2]:

$$L_s(k) = \frac{1}{k} \sum_{r=1}^{k} B(r)$$

Theorem 2. The expected number of additional block accesses for an unsuccessful search when the length of the overflow chain is k objects is given by [2]:

$$L_u(k) = \frac{1}{k+1} \left(\sum_{r=1}^{k} B(r) + B(k) \right)$$

3.2. Second algorithm

Theorem 3. The expected number of additional block accesses for the successful search of any object out of the k ones of an overflow chain is:

$$L_s(k) = \frac{1}{k} \left(\sum_{r=1}^{k} \left[\sum_{i=1}^{r-1} B(i) \left(\frac{B(i)b - i}{N_{ov}(a) - i} - \frac{B(i-1)b - i + 1}{N_{ov}(a) - i + 1} \right) + B(r) \left(1 - \frac{B(r-1)b - r + 1}{N_{ov}(a) - r + 1} \right) \right] \right)$$

Proof. Assuming that k objects constitute the overflow chain then this formula is derived using an expected value analysis as follows. The probability of finding the object that we are looking for (e.g. the rth one) in any of the $B(i)$ blocks that we have in main memory when we examine the block with the ith object ($i < r$) in the chain is: $(B(i)b - i)/(N_{ov}(a) - i)$. This is true because when the rth object is examined, $B(i)b$ objects are cached and i of them are not relevant. The probability of finding this object exactly at the search for the ith object and not before is: $(B(i)b - i)/(N_{ov}(a) - 1) - (B(i-1)b - i + 1)/(N_{ov}(a) - i + 1)$. The probability that the rth object is not found in any of the blocks which contain the $r - 1$ first b objects in the chain is: $(B(r-1)b - r + 1)/(N_{ov}(a) - r + 1)$. The above formula follows easily. □

3.3. Third algorithm

The analysis of this algorithm is more complex and can be formulated using non-homogeneous Markov chains. Define the ith state to indicate that i block fetches have already taken place. The state i is characterized by a one-dimensional vector Π_i, where $i = 1, 2, \ldots, k$. The vector length is k, the length of the overflow chain. The jth element of the vector represents the probability that the object with the highest key in this block is the jth object (out of the k) of the overflow chain.

A transition happens when a new block is fetched. By definition, the ith transition happens when the $(i + 1)$th block is going to be fetched, where $i = 1, \ldots, k - 1$. A transition is characterized by a two-dimensional translation matrix P_i of size $k \times k$. The element $P_i(p, q)$, where $1 \leqslant p \leqslant k - 1$ and $2 \leqslant q \leqslant k$, represents the probability that, if at the ith state the object with the highest key value is the pth object of the overflow chain, then the object with the highest key value after the ith transition will be the qth object. The aspect that makes it non-homogeneous is that the transition matrix changes after each transition.

Theorem 4. The probability that exactly i objects of the chain exist in the block, in which the first object of the chain resides, is equal to:

$$r(i, N_{ov}(a), k) = \frac{\binom{b-1}{i-1}\binom{N_{ov}(a)}{k-i}}{\binom{N_{ov}(a)-1}{k-1}}$$

where $N_{ov}(a)$ denotes the total number of overflow objects in area and k denotes the length of the chain.

Proof. The numerator gives the number of ways that the $i - 1$ objects of the chain may be selected from the $b - 1$ objects of a specific block (it is certain that the first object of the chain resides also in this block) multiplied by the number of the ways that the rest $k - i$ objects of the chain may be selected from the rest $N_{ov}(a) - b$ objects. The denominator is self-explained. □

Theorem 5. Given that i objects out of k ones of the chain reside in the block, where the first object of the chain is, then the probability, that the order of the one that has the maximum order in the chain is j, is equal to:

$$s(j, i, k) = \frac{\binom{j-2}{i-2}}{\binom{k-1}{i-1}}$$

Proof. The explanation of $s(j, i, k)$ is the following. In a specific block reside i objects; two of them are the first and the jth objects of the chain. Therefore the numerator gives the number of ways that $i - 2$ objects may be selected from the $j - 2$ ones. To calculate the denominator consider that the first object of the chain is fixed, therefore the rest $i - 1$ objects of the chain may be selected from the rest $k - 1$ objects of the chain. □

Theorem 6. The initial vector Π_i is given by:

$$\Pi_i(j) = \sum_{i=2}^{k} r(i, N_{ov}(a), k)s(j, i, k) \quad \text{where } 2 \leqslant j \leqslant k$$

$$\Pi_i(1) = 1 - \sum_{r=2}^{k} \Pi_i(r)$$

Proof. The above relation is explained easily by considering the theorems deriving the expressions $r(i, N_{ov}(a), k)$ and $s(j, i, k)$. Note, also, that the upper bound in the summations of the previous formula may be replaced by the expression $min(b, j)$. This is true because values between this expression and the chain length k do not contribute to the final result, since the combination produced by $r(\cdot)$ and $s(\cdot)$ are equal to zero. □

Theorem 7. The transition matrix P_m, where $m = 1, \ldots, k-1$, is:

$$P_m(p, q) = 0 \qquad \text{for } q \leqslant p \text{ or } p < m$$

$$P_m(p, q) = \sum_{i=2}^{k-p} r(i, N_{ov}(a) - mb, k-p)\, s(q-p, i, k-p) \qquad \text{for } q \geqslant p+2$$

$$P_m(p, p+1) = 1 - \sum_{q=p+2}^{k} P_m(p, q)$$

Proof. The explanation follows in two steps. The probability that before some block access the object with the highest key value is the pth, while after the block access the object with the highest key value in the overflow chain is the qth one, where $q \leqslant p$, is evidently zero. In case $m > p$, then again this probability is equal to zero. This is due to the fact that the order of the transition (block access) has to be less or equal to the order of the object which has the highest key value so far.

The explanation of the second and the third part of the previous relation is based on a reasoning similar to that of Theorem 6. Assume that m blocks have been retrieved already, while the object with the highest order so far is the pth one. The number of objects contained in the rest blocks is: $N_{ov}(a) - bm$. Some of the first p objects may be among these $N_{ov}(a) - bm$ objects but they are considered as irrelevant in the process and, therefore, we concentrate on the rest $k-p$ objects of the chain. When the $(m+1)$th block is retrieved it contains i objects out of $k-p$ ones, where i varies from 1 up to $k-p$. The object with the highest order out of these i objects is the jth of the chain with length k but now it is considered to be the $(q-p)$th one of the rest of the chain. Therefore, by substituting $N_{ov}(a)$ with $N_{ov}(a) - bm$, k with $k-p$ and j with $q-p$ and summing over i we finally come up with the formula of the theorem. □

Note, again, that following the reasoning of Theorem 6 the upper bound in the summations of the formulae of the Theorem 7 may be replaced by the expression $min(b, q-p)$ because values between this expression and the chain length $q-p$ do not contribute to the final result. If this formula m is replaced by 1, then we get the first transition matrix (which is easily checked), while if we replace m with 0 then the initial probability vector of Theorem 6 is derived.

The time dependent state probabilities after the ith transition are given by Kleinrock [19]:

$$\Pi_{i+1} = \Pi_i P_i = \Pi_1 P_1 P_2 \ldots P_i$$

In the Appendix a practical example validating the above analysis for the successful search of the third algorithm is given.

Theorem 8. The expected number of additional block accesses for the successful search of any object out of the k ones of an overflow chain is:

$$L_s(k) = \frac{1}{k} \sum_{j=1}^{k} \sum_{i=1}^{j} i\, \Pi_i(j)$$

Proof. The expected value of the additional accesses for the successful search of the last (kth) object of a chain is: $\Sigma_{i=1}^{k} i\Pi_i(k)$. This formula is explained easily by considering the product of the ith state probability, $\Pi_i(k)$, by the relevant cost (i block accesses). It is reminded again that the ith state probability denotes the probability that the last object of the chain (the kth one) will be found after the ith transition, where $i = 1, 2, \ldots, k-1$, (e.g. at the retrieval of the $(i+1)$th block). It is understood that the expected additional cost for the successful search of any object of the chain (e.g. the ith one, where $1 \leqslant i < k$) is calculated by considering that this object is the last one. For example, suppose that $k = 3$, which means that three chained overflow objects may be stored in 1, 2 or 3 blocks. The access cost of the first object is: $1 \times \Pi_1(1)$, which evidently equals 1 block access. The second object may be found after 1 or 2 block accesses, therefore the access cost of the second object is: $1 \times \Pi_1(2) + 2 \times \Pi_2(2)$. The third object may be found after 1, 2 or 3 block accesses, thus the access cost of the third object is: $1 \times \Pi_1(3) + 2 \times \Pi_2(3) + 3 \times \Pi_3(3)$. The summation gives the total successful cost, while dividing by 3 we get the mean successful cost. By generalizing this reasoning we get the formula of the theorem. □

Theorem 9. The expected number of additional block accesses for an unsuccessful search when the length of the overflow chain is k objects is:

$$L_u(k) = \frac{1}{k+1}\left(\sum_{j=1}^{k} \sum_{i=1}^{j} i\Pi_i(j) + 1 + \sum_{j=2}^{k} \sum_{i=2}^{k} \Pi_i(j) \right)$$

Proof. The unsuccessful search analysis is based on that of the previous theorem. Suppose that the chain consists of k objects, therefore $k+1$ subintervals are created. It is certain that absence of an object is recognized by identifying the two consecutive objects of the chain which have smaller and greater key value respectively than the key value of the desired one. The following example explains how the unsuccessful search cost is derived.

Suppose that $k = 3$, which means that 3 chained overflow objects may be stored in 1, 2 or 3 blocks. However in this case 4 subintervals are created and therefore we have to consider the excess cost of 4 nonexisting objects. To guarantee that an object does not exist, we have to perform a transition from an object with a key value smaller than the key value of the desired object to an object with a key value greater than the key value of the desired one. Table 2 shows the relevant costs for 4 nonexisting objects.

The first column of Table 2 gives the cost for accessing the object with the greatest key value which is smaller than the key value of the desired object. The second column gives the cost paid to access the next object of the chain given that the previous one has already been accessed. It is evident that in general the latter cost is smaller than the unity because it has to be paid only in cases where the next object does not reside in one of the previously accessed blocks. More specifically:

- the cost for the first subinterval equals unity, and
- the cost for the last subinterval equals zero because no other block may be accessed.

Another two interesting observations are the following ones:

- the values of the first and second column form a incrementing and decrementing sequence respectively and
- except the first and last subintervals, in each line an equal number of products is summed.

By summing we get the total cost and dividing by four we get the mean unsuccessful cost. By generalizing we get the formula of the theorem and assuming that every file object has the same probability to be accessed, then the following theorem holds. □

Theorem 10. The expected number of additional block accesses for a successful search or an unsuccessful search respectively is:

$$S(a) = \frac{1}{X(a)} \sum_{x=2}^{\infty} P(x,a) \sum_{k=1}^{\infty} kL_s(k)Q(x-k,x)$$

$$U(a) = \frac{1}{X(a)} \sum_{x=2}^{\infty} P(x,a) \sum_{k=1}^{\infty} kL_u(k)Q(x-k,x)$$

where $X(a)$ is the expected number of objects per area and equals: $\Sigma_{x=1}^{\infty} xP(x,a)$.

Proof. The explanation is as follows. $Q(x-k,x)$ is the probability that there are exactly $x-k$ objects in a main block, and therefore at the same time it expresses the probability that there are exactly k overflows. When there are k overflows in an area the expected additional accesses in order to find an object is: $kL_s(k)Q(x-k,x)$. Finally the probability of accessing a block which contains x objects is: $xP(x,a)/X(a)$. Substitution results in the first relation of the theorem. Similar is the explanation of the second part of the theorem. □

Table 2

	Cost for accessing the previous object		Cost for accessing the next object
1st subinterval			$1 \times \Pi_1(1)$
2nd subinterval	$1 \times \Pi_1(1)$	+	$1 \times \Pi_2(2)$
3rd subinterval	$1 \times \Pi_1(2) + 2 \times \Pi_2(2)$	+	$1 \times \Pi_2(3) + 1 \times \Pi_3(3)$
4th subinterval	$1 \times \Pi_1(3) + 2 \times \Pi_2(3) + 3 \times \Pi_3(3)$		

Corollary. More efficient computationally formulae may be applied in place of the formulae of Theorem 10, such as:

$$S(a) = \frac{1}{X(a)} \sum_{k=1}^{\infty} kL_s(k) \sum_{x=k+1}^{\infty} P(x, a)Q(x-k, x)$$

$$U(a) = \frac{1}{X(a)} \sum_{k=1}^{\infty} (k+1)L_u(k) \sum_{x=k+1}^{\infty} P(x, a)Q(x-k, x)$$

These relations are more efficient because the time consuming $L_s(k)$ and $L_u(k)$ are computed only at the outer loop. The derivation of these relations is straightforward.

4. PERFORMANCE COMPARISONS

In this section numerical results are presented using the analytical estimates derived in the previous sections. These results are used to compare the performance of the three overflow search algorithms as well as to study the effect of variable length objects on performance. An index sequential file is considered in our experiments, therefore the relation of Lemma 2 is used as the probability distribution for estimating the number of objects per area. Similar results may be obtained for hash based files. In all experiments.

- the main and overflow block size is 4 kbytes
- the object classes are two and
- the object size of the first class is always 400 bytes and
- every block initially is loaded with 7 objects.

The object size of the second class takes a value from the set of 400, 300 or 200 bytes, therefore the load factor varies from 70 to 50%.

All the figures show the additional successful and unsuccessful expected cost as a function of the file expansion factor. In Figs 3–7, 8–12 and 13–17, the parameter *MB* of the number of main blocks which share some overflow blocks equals 1000, 100 and 10 blocks, respectively. In every figure five lines depict the result of plugging the formulae of the Theorems 1, 2, 3, 8, 9 as appropriate

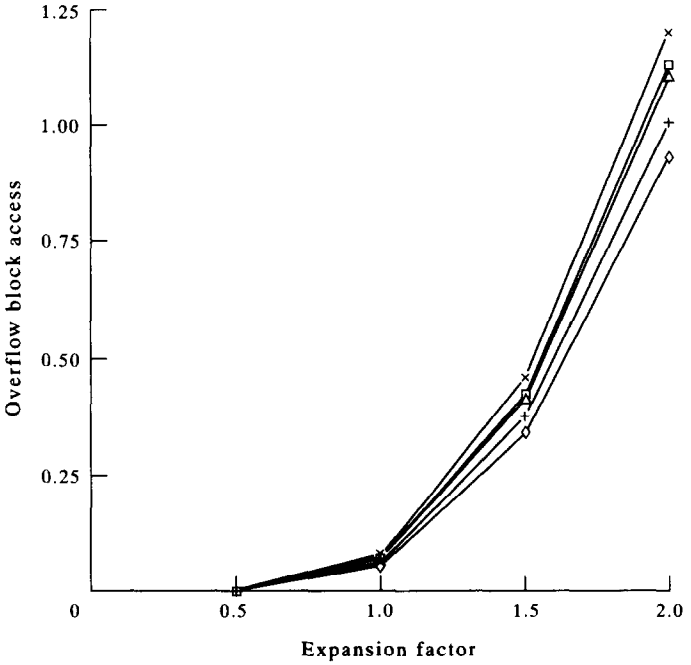

Fig. 3. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 200$ b, $P_1 = 0.1$, $P_2 = 0.9$, $MB = 1000$.

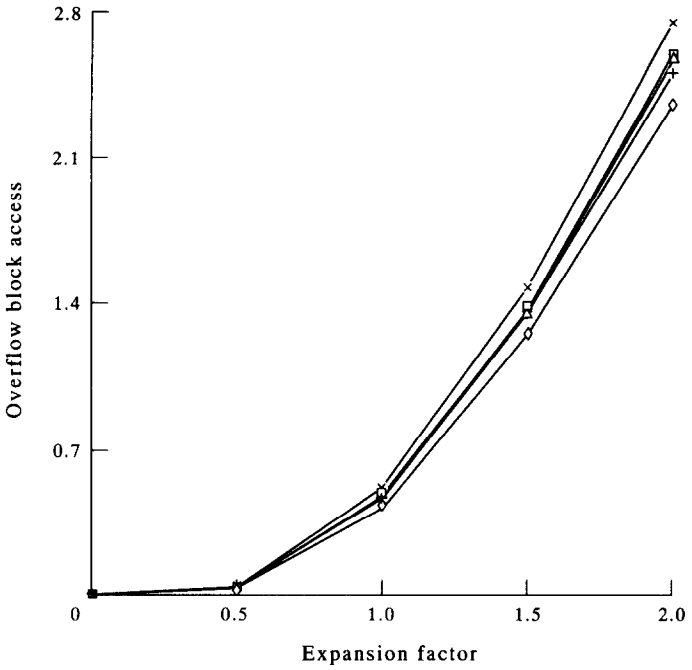

Fig. 4. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 200$ b, $P_1 = 0.5$, $P_2 = 0.5$, $MB = 1000$.

into the relevant formulae of the corollary of the previous section. The values of all other parameters is given below each figure.

In the beginning, let us note some expected conclusions. First, in any case unsuccessful searching is more expensive than successful searching. Second, the third algorithm is always better than the second one, which in turn is better than the first one. In all figures, the unsuccessful search cost of the third algorithm is smaller than the successful search cost of the first and the second algorithms.

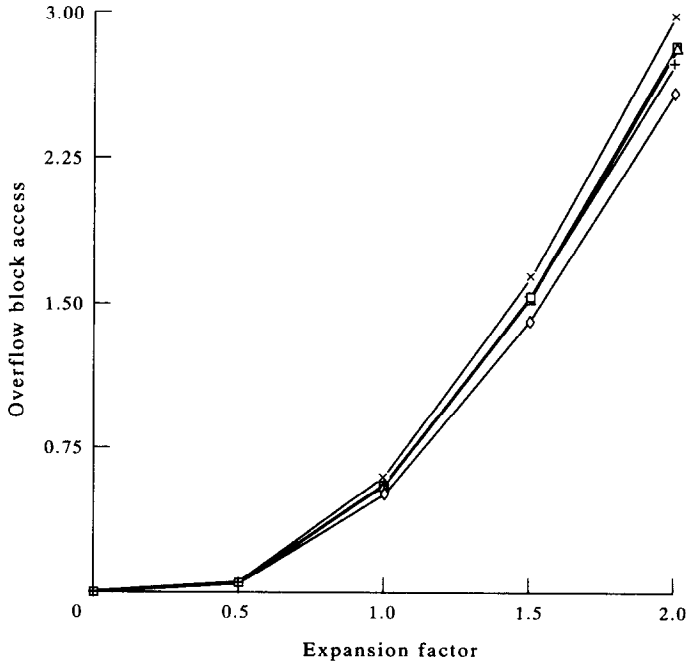

Fig. 5. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 300$ b, $P_1 = 0.1$, $P_2 = 0.9$, $MB = 1000$.

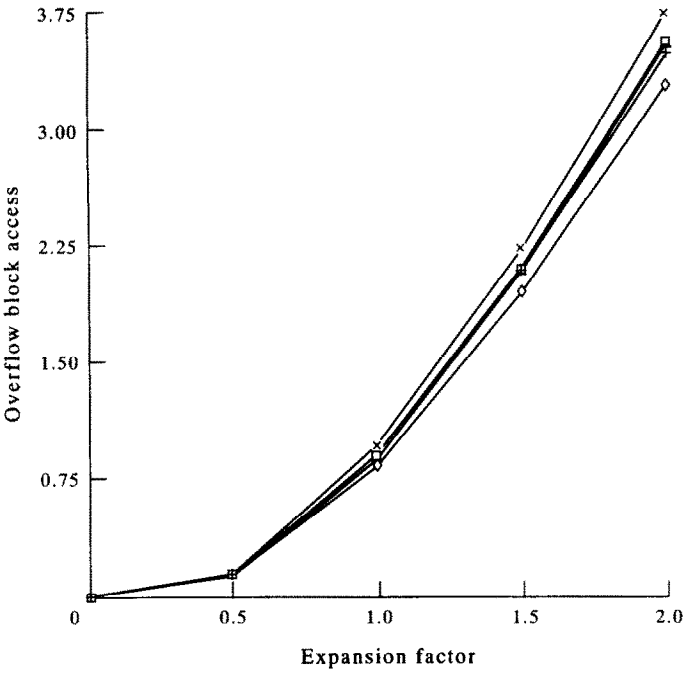

Fig. 6. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 300$ b, $P_1 = 0.5$, $P_2 = 0.5$, $MB = 1000$.

Two important conclusions of this study are the following. First, it is remarked that as the number of main blocks MB pointing to the same overflow area increases, the first two algorithms tend to behave similarly, while the third one still shows a remarkable improvement. When the parameter MB is small there is an evident improvement when comparing the second to the first algorithm, while there is an outstanding improvement when comparing the third to the first algorithm. Second, it is remarked that the gain due to the third algorithm increases by increasing the probability of the short object size or by decreasing the short object size. This is the same with

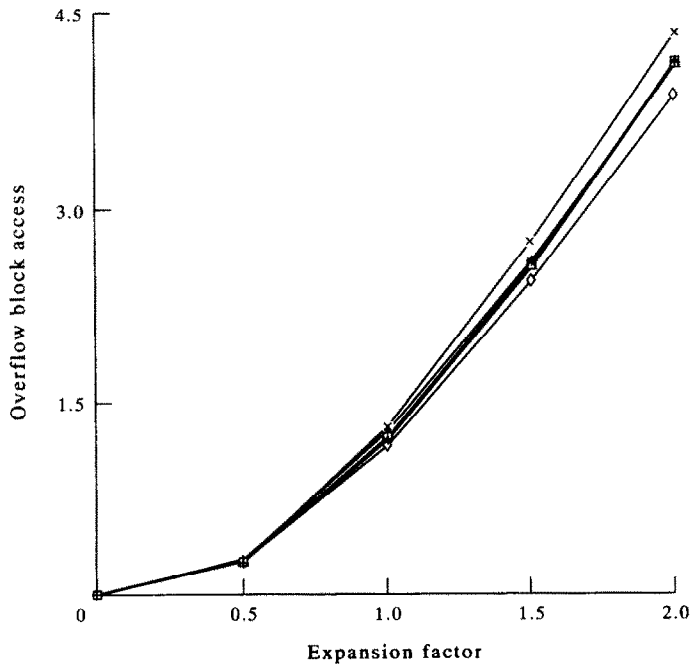

Fig. 7. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 400$ b, MB = 1000.

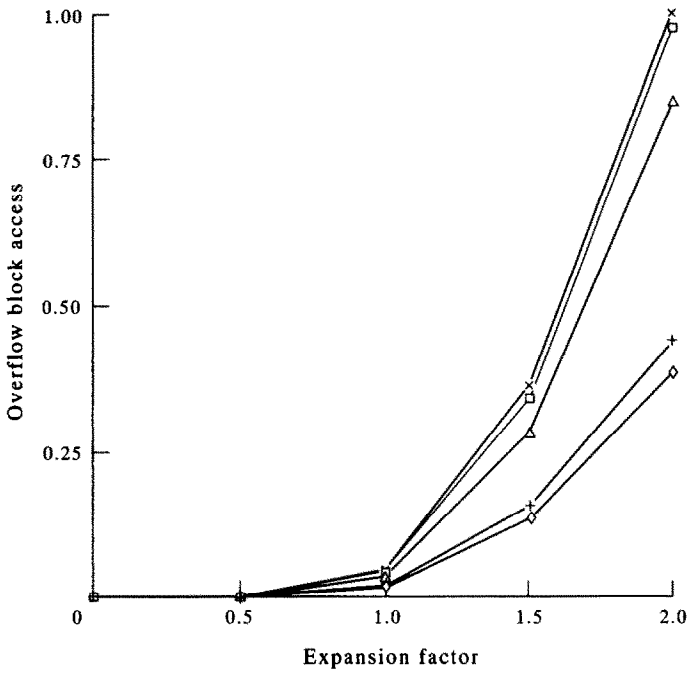

Fig. 8. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 200$ b, $P_1 = 0.1$, $P_2 = 0.9$, $MB = 1000$.

stating that the gain due to the third algorithm increases by decreasing the average object size compared to the block size.

The above observations are explained as follows. There are three factors affecting the expected cost. The first is the average object length. The smaller the average object length is the more objects are needed to fill the main block and therefore the less expected number of blocks in the overflow chain. The second factor is the empty space left at the end of the main block. The larger the size of the objects (with respect to the block size), the more will be the wasted space at the end of the

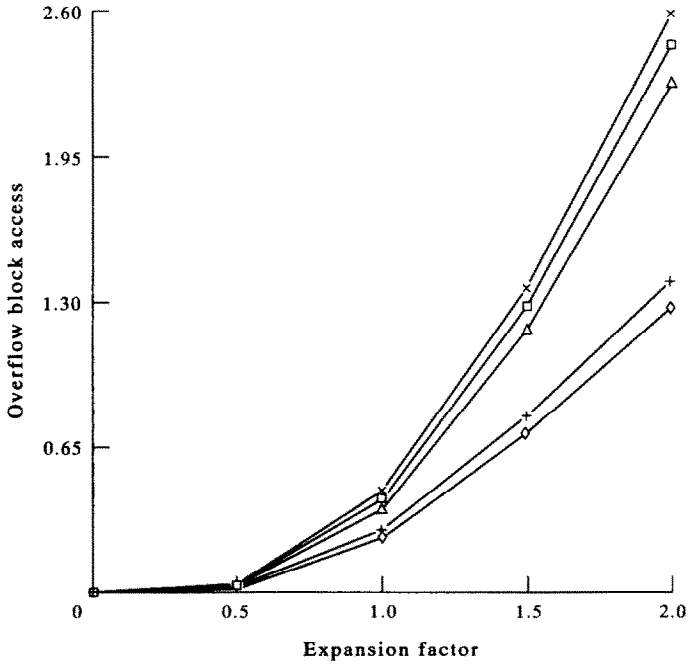

Fig. 9. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 200$ b, $P_1 = 0.5$, $P_2 = 0.5$, $MB = 100$.

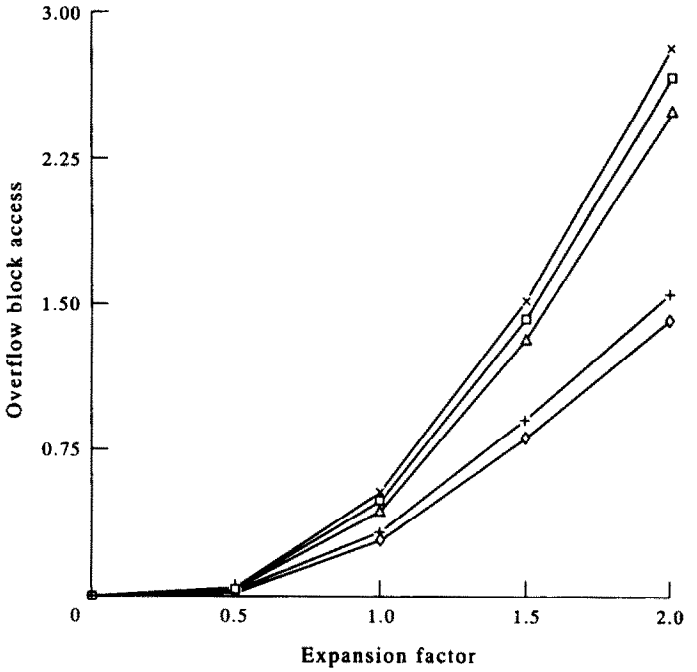

Fig. 10. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 300$ b, $P_1 = 0.1$, $P_2 = 0.9$, $MB = 100$.

main block [6]. A third factor is the fact that the larger number of objects per overflow block results in higher probability of finding more overflow objects of the same area in one overflow block.

5. SUMMARY—CONCLUSION

In this paper we have studied a file organization with shared overflow blocks. The performance of two new algorithms for searching in the overflow file blocks for variable length objects is analyzed. We have derived estimates of their performance costs taking into account the statistical

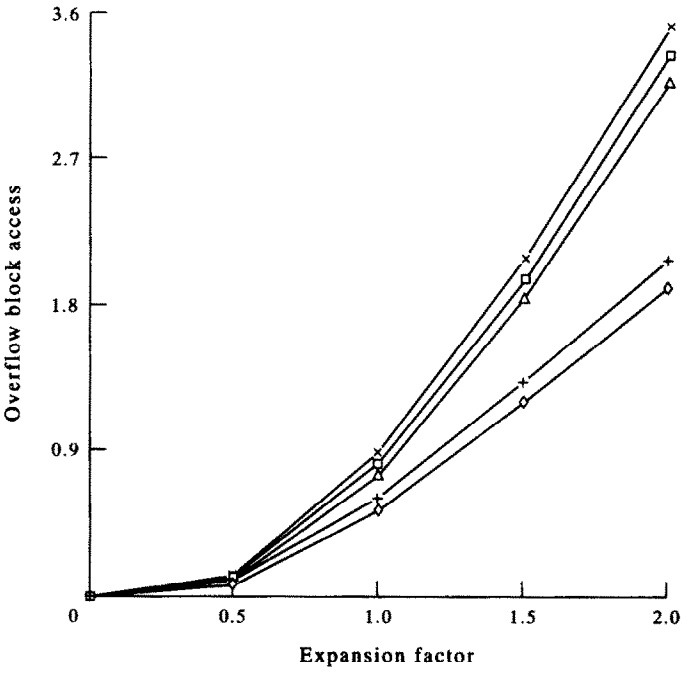

Fig. 11. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 300$ b, $P_1 = 0.5$, $P_2 = 0.5$, $MB = 100$.

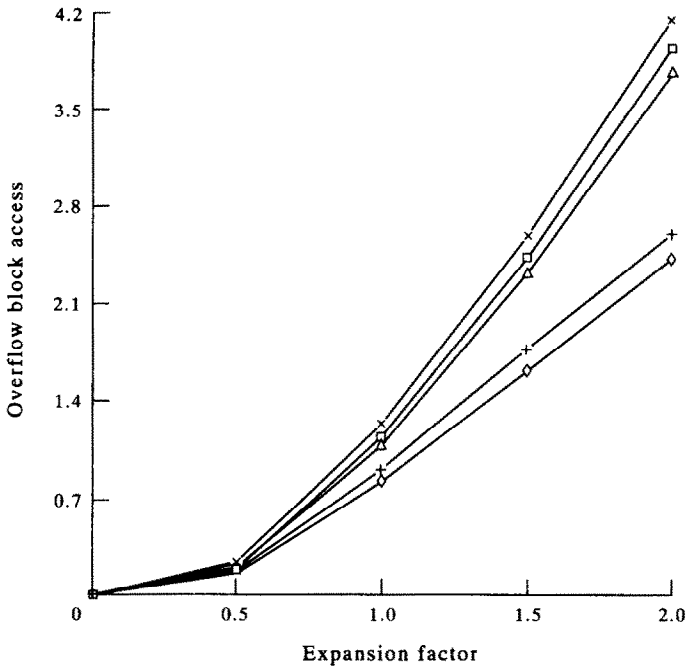

Fig. 12. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 400$ b, MB = 100.

probability distribution of the object lengths of the underlying population of objects as well as other file parameters. It is proved that both the new algorithms outperform the previous one. Specifically, the most sophisticated one presents remarkable performance improvement in the following two cases:

- the objects have great variance in their lengths,
- the object sizes are relatively small when compared with the block size, and
- the number of main blocks that share a group of some overflow blocks is not great.

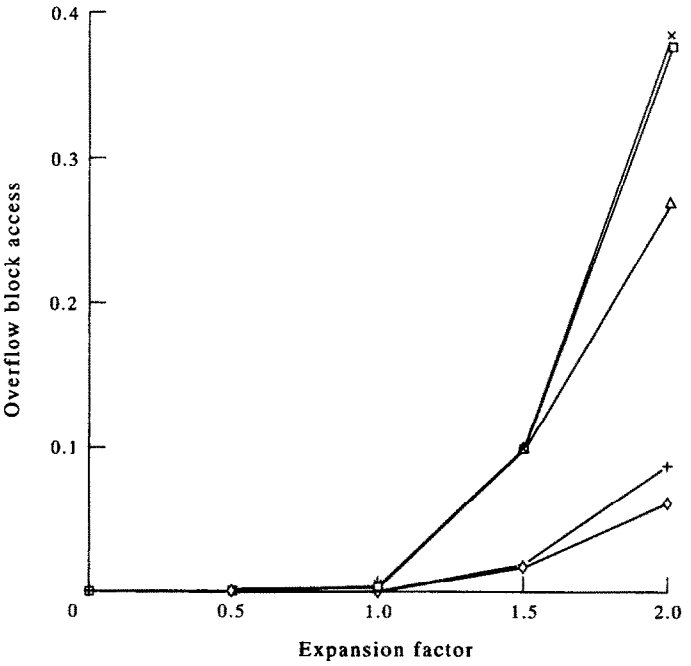

Fig. 13. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 200$ b, $P_1 = 0.1$, $P_2 = 0.9$, $MB = 10$.

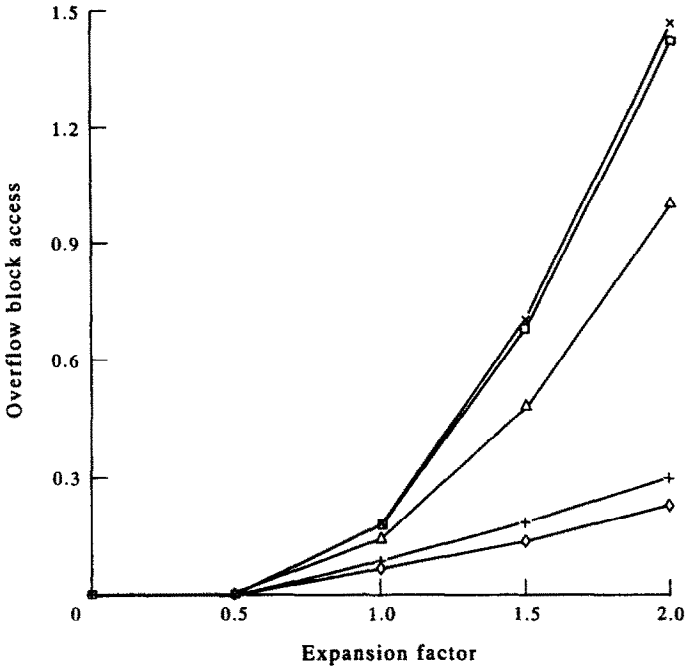

Fig. 14. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 200$ b, $P_1 = 0.5$, $P_2 = 0.5$, $MB = 10$.

Based on the above observations it is concluded that the file structure with overflows managed as described in the second chapter, benefits both from the search algorithms described and analyzed in the third chapter, as well as from the additional clustering of overflow objects. According to this technique a small number of overflow areas is maintained instead of a single one. In addition this small number of overflow areas (each for a group of main blocks) achieves very good space utilization.

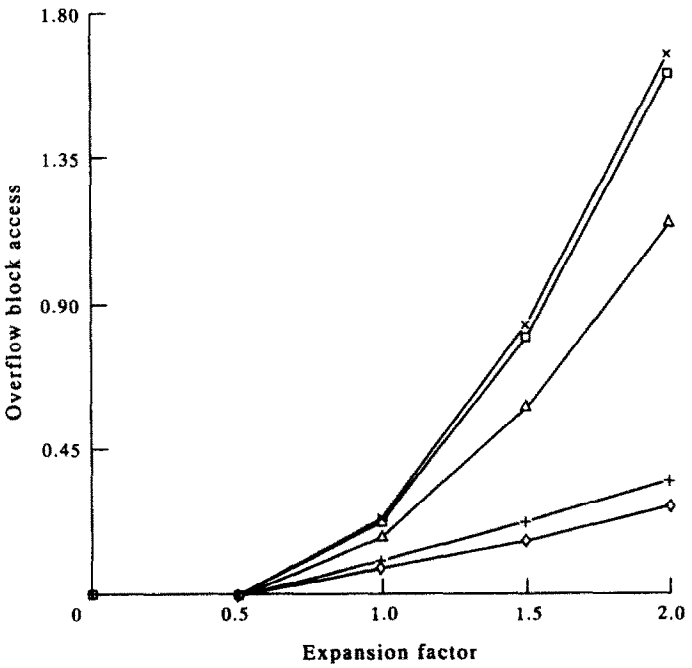

Fig. 15. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 300$ b, $P_1 = 0.1$, $P_2 = 0.9$, $MB = 10$.

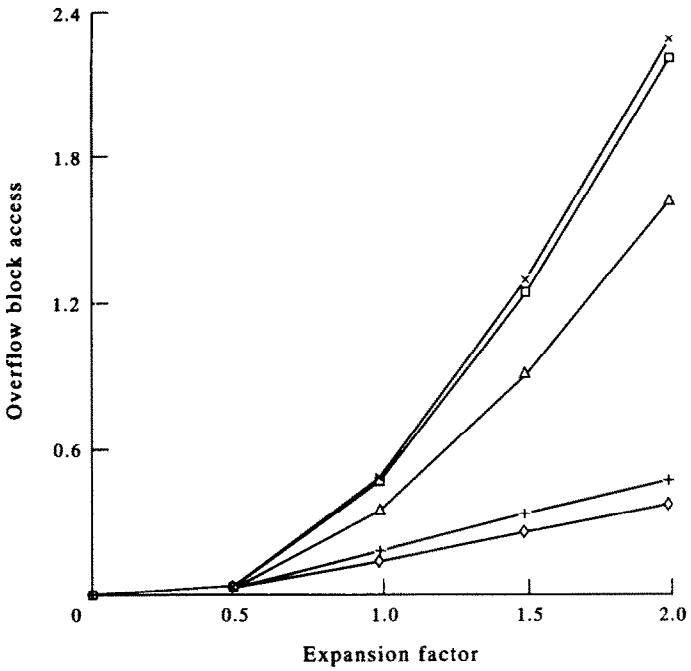

Fig. 16. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 300$ b, $P_1 = 0.5$, $P_2 = 0.5$, $MB = 10$.

Future research in this area involves the derivation of some approximations of the cost equations presented in order to obtain a less analytical model. In addition this overflow scheme and the corresponding algorithms would be tested in a hashed based file. Another step would be towards the development of structures and algorithms designed specially for variable length objects [20].

Acknowledgements—Thanks are due to Mr D. Meimaris and Mr G. Priovolos for their help in experimentation. A reviewer's and Mr K. Dobrolis's comments on the presentation of the paper are appreciated.

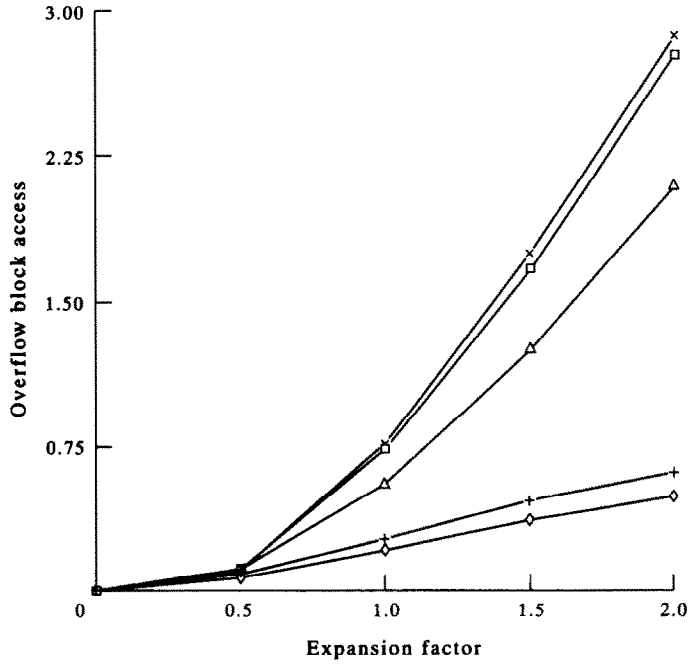

Fig. 17. Search cost as a function of the file expansion factor. Parameters: $BS = 4$ kb, $R_1 = 400$ b, $R_2 = 200$ b, MB = 10.

REFERENCES

[1] R. J. Enbody and H. C. Du. Dynamic hashing. *ACM Comput. Surv.* **20,** 85–113 (1988).
[2] S. Christodoulakis, Y. Manolopoulos and P. A. Larson. Analysis of overflow handling for variable length records. *Information Systems* **14,** 151–162 (1989).
[3] P. A. Larson. Analysis of index sequential files with overflow chaining. *ACM Transact. Database Syst.* **6,** 671–680 (1981).
[4] J. Hakola and A. Heiskanen. On the distribution of wasted space at the end of file blocks. *BIT* **20,** 145–156 (1980).
[5] G. U. Hubbard. *Computer-assisted Database Design.* Van Nostrand Reinhold (1981).
[6] Y. Manolopoulos and C. Faloutsos. Analysis for the end of block wasted space. *BIT* **12,** 620–630 (1990).
[7] T. J. Teorey and J. P. Fry. *Design of Database Structures.* Prentice-Hall, NJ (1982).
[8] G. Wiederhold. *File Organization for Database Design.* McGraw-Hill, NY (1987).
[9] M. J. Willshire. How spacey can they get? Space overhead for storage and indexing with object-oriented databases. In *Proc. 7th IEEE Data Engineering Conf.*, pp. 14–22 (1991).
[10] D. H. Fishman, D. Beech, H. P. Cate, E. C. Chow, T. Connors, J. W. Davis, N. Derrett, C. G. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M. A. Neimat, T. A. Ryan and M. C. Shan. IRIS: An object-oriented database management system. *ACM Transact. Office Inf. Syst.* **5,** 48–69 (1987).
[11] J. Banerjee, H. T. Chou, J. F. Garza, W. Kim, D. Woelk, N. Ballou and H. J. Kim. Data models issues for object-oriented applications. *ACM Transact. Office Inf. Syst.* **5,** 3–26 (1987).
[12] J. Mylopoulos, P. A. Bernstein and H. K.T. A language facility for desiging database intensive applications. *ACM Transact. Database Syst.* **5,** 185–207 (1980).
[13] D. Batory. Optimal file designs and reorganization points. *ACM Transact. Database Syst.* **7,** 60–81 (1982).
[14] O. Amble and D. E. Knuth. Ordered hash tables. *Comput. J.* **17,** 135–147 (1974).
[15] G. H. Gonnet and P. A. Larson. External hashing with limited internal storage. *J. ACM* **35,** 161–184 (1988).
[16] T. S. Yuen and D. H. C. Du. Dynamic file structure for partial match retrieval based on overflow bucket sharing. *IEEE Transact. Software Engng* **12,** 801–810 (1986).
[17] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. *ACM Transact. Database Syst.* **9,** 163–186 (1984).
[18] S. Christodoulakis. Estimating block transfers and join sizes. In *Proc. ACM SIGMOD-83 Conf.*, pp. 40–54 (1983).
[19] L. Kleinrock. *Queueing Systems, Vol. 1: Theory.* Wiley, NY (1975).
[20] Y. Manolopoulos and N. Fistas. Algorithms for hash based files with variable length records. *Information Sciences.* In press (1992).

APPENDIX

Here, a simple example is given showing how the analysis is applied. Suppose that a group of shared overflow blocks corresponds per $MB = 10$ main blocks of size 1000 bytes. Two object types are considered: the first length 400 bytes and probability 0.5 and the second of length 300 bytes and probability 0.5. Suppose also that the last object of a chain of 4 objects is to be searched. By applying the analysis of Sections 2 and 3 it is derived that:

$$\Pi_1 = [0{,}8859 \quad 0{,}0380 \quad 0{,}0380 \quad 0{,}0380]$$

$$P_1 = \begin{bmatrix} 0 & 0{,}9176 & 0{,}0412 & 0{,}0412 \\ 0 & 0 & 0{,}9583 & 0{,}0417 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Pi_2 = [0 \quad 0{,}8129 \quad 0{,}0745 \quad 0{,}0745]$$

$$P_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}9551 & 0{,}0449 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Pi_3 = [0 \quad 0 \quad 0{,}7764 \quad 0{,}0110]$$

$$P_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Pi_4 = [0 \quad 0 \quad 0 \quad 0{,}7764]$$

Three interesting observations about these results follow.

- The elements of the jth line of the transition matrix sum to unity if the $j > i - 1$, where i is the state of block fetches, otherwise they are zeros.
- The sum of the last elements of all the probability vectors is equal to one. This is explained by the fact that the last object will certainly be found in one out of the four blocks.
- The sum of the elements of Π_1 is equal to one. This is expected because it is certain that one out of the four objects will be the found in the first block. This is, also, the explanation why the elements of the other probability vectors Π_i, $i = 2, 3, \ldots, k$ do not sum to unity.