Pergamon

Information Systems Vol. 19, No. 5, pp. 433–446, 1994
Copyright© 1994 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0306-4379/94 $7.00 + 0.00

0306-4379(94)00021-2

PERFORMANCE OF LINEAR HASHING SCHEMES FOR PRIMARY KEY RETRIEVAL†

Y. MANOLOPOULOS[1‡] and N. LORENTZOS[2]

[1]Computer Science Department, University of Maryland
College Park, MD 20742, USA

[2]Informatics Laboratory, Agricultural University of Athens
Athens, 11855 Greece

(Received 3 August 1993; in final revised form 7 April 1994)

Abstract — Linear hashing is one of the most attractive dynamic hashing schemes. Linear hashing with partial expansions and linear hashing with priority splitting are two variations with improved space and time performance. Here, we propose a new structure, which is termed *linear hashing with partial expansions and priority splitting.* The above four structures are compared by simulation and it is shown that the new scheme outperforms its predecessors in both time and space costs.

1. INTRODUCTION

Static hashing schemes (i.e. open addressing and separate/coalesced chaining) suffer because of space waste and/or time performance deterioration. During the last decade many *dynamic hashing schemes* were proposed. These file organizations grow and shrink due to insertions and deletions respectively [16]. Therefore the time performance of searches, inserts, and deletes is essentially independent of file size. Some examples of dynamic hashing schemes are *dynamic hashing* [3], *linear hashing* [10], *virtual hashing* [9], *extendible hashing* [2], and *spiral hashing* [12]. Each organization from this incomplete list has been studied extensively in [1].

Linear hashing and spiral hashing have a major advantage over their counterparts; they do not make use of an index structure, which may require so much space, so as to not fit in main memory. In some other hashed organizations, this structure might have to be stored in secondary storage and consequently might result in costly disk access operations. In addition, spiral hashing is characterized by another major drawback when compared with linear hashing, i.e. the complicated space allocation algorithms.

Because of these advantages, linear hashing has been investigated exhaustively so as to maximize its performance. Numerous variations have been proposed which incorporate linear hashing for primary key access organizations, such as [4, 5, 6, 7, 8, 13, 14, 15]. In addition, linear hashing schemes for multidimensional searching or for partial match retrieval with/without accommodating the order preserving property were recently designed and proposed. Enhancements of these types are not discussed in the present paper but the interested reader may find citations to the most important variations of this category in [1].

The present paper focuses on the following primary key retrieval organizations: The original *Linear Hashing (LH)* [10], *Linear Hashing with Partial Expansions (LHPE)* [4, 5] and *Linear Hashing with Priority Splitting (LHPS)* [15]. In addition, a new variation is proposed, which extends all previous approaches. We call it **Linear Hashing with Partial Expansions and Priority Splitting (LHPEPS)** and compare it with its predecessors. The remainder of this work is divided as follows: The LH, LHPE and LHPS schemes are presented in Section 2. The LHPEPS scheme is described in Section 3. In Section 4 the performance of LH, LHPE, LHPS, and LHPEPS is compared by simulation. Conclusions are drawn in the last section.

†Recommended by P. O'Neil.

‡On sabbatical leave from the Department of Informatics, Aristotle University, Thessaloniki, 54006 Greece

2. LINEAR HASHING AND EXTENSIONS

Linear hashing schemes for primary key retrieval can be classified in two types: Organizations which store the overflow records in a separate file, and those which accommodate the overflow records in the main file [6, 7, 8, 13]. The latter will not be examined in the sequel. The structure of *Recursive Linear Hashing*, which is characterized by many overflow files, will not be examined either [14].

We start by the presentation of the LH scheme. To this end, assume that initially the file consists of $N_0 \times 2^0$ ($=N_0$) records, where the power of 2 represents the *level* of the file expansion. Hence, the file is initially at the 0-th level. A hashing function, such as '$h_1(key) = key \ mod \ N_0$', is then used to store and/or retrieve records. It should be noted that *key* is the suffix of the binary representation of the key, whose length equals the level of file expansion (initially 0). If the file is overloaded to the extent that the storage utilization factor exceeds a predetermined threshold value, then the file expands by splitting the 0-th main block out of the N_0 ones, thus creating a new main block with address N_0. At the same time, the records of the original 0-th block are redistributed in the 0-th and the N_0-th block, according to a new hashing function, '$h_2(key) = key \ mod \ 2N_0$', where *key* now denotes a binary suffix of length $l+1$ (i.e. 1 when the first split occurs). In this way the storage utilization factor becomes smaller than the predetermined value. Whenever later on, the threshold value is reached again, the second main block (address 1) splits in the same way, thus creating a new address (address $N_0 + 1$) and the records are redistributed accordingly, using the second hashing function.

As is obvious, this expansion process is repeated by splitting the main blocks of the original file, one at a time, until their total number is doubled. At that moment the first expansion has been completed. As it is also obvious, a pointer, 'next', is necessary to give the address of the main block which is going to split next.

During this expansion some main blocks are accessed via the first hashing function ($h_1(key)$), whereas some others are accessed via the second one ($h_2(key)$). The necessity to use two hashing functions, is a direct consequence of the lack of an index structure. The decision concerning the function which has to be used is taken as follows: During a cycle, pointer 'next' equals the 'boundary value', i.e. the address of the first main block which should be accessed via the second hash function. If the result of the first hashing function is (not) smaller than the boundary value then the (first) second hashing function has really to be used.

At the end of the first cycle, the file consists of $2N_0$ main blocks and only the second hashing function ($h_2(key)$) is necessary. When the second cycle begins, pointer 'next' is set to the 0-th main block, the first one to split. During the second cycle this pointer's value increases linearly, set each time to the value of the block which will split next (up to block ($2N_0 - 1$)) and a new hashing function, '$h_3(key) = key \ mod \ 4N_0$', is being used, where *key* is a binary suffix of length 2. From that point on, the process is repeated in a similar way.

Best performance is achieved if the keys are uniformly distributed in blocks. However, in practice there is high probability for a block to have overflow records. Such records are stored in a separate overflow file, whose blocks have a different (usually smaller) capacity than that of the main file blocks. In addition, only one chain of overflow blocks may emanate from each main block. It is worth noting that the decision concerning the block which is to split next and when this split will take place does not take into consideration overflow records in some particular block. It should also be noted that the main file blocks are split one at a time in a linear fashion. As a consequence, long overflow chains may exist for long time intervals. This has two effects on the performance behavior of LH:

- Space utilization varies substantially from one block to another. It is very probable that the storage utilization factor of the blocks that have already split is approximately 50%, whereas this factor is nearly 100% for the blocks which have not split yet.

- Overflowing causes time performance deterioration. Search and insert operations are more costly because they have to access a chain of blocks.

The goal of each of the variations, which are described next, is to distribute records more evenly over the file.

The LHPE scheme aims at overcoming both of the disadvantages described above and, indeed, this is achieved to some extent. The main difference when compared with the LH scheme is the following: Assume that the file consists of N_0 main blocks, where N_0 is an even number. Whenever it is determined that an expansion has to take place, the i-th and the $(i + N_0/2 - 1)$-th block are involved in a two-to-three split. After all such pairs have been processed in this way, the file has $3N_0/2$ blocks and we say that a 'partial expansion' has taken place. The next step is to combine blocks in a three-to-four split: The indices of the three blocks to be involved in the split, are i, $i + N_0/2$ and $i + N_0$, where $0 \leq i \leq N_0/2 - 1$. Thus, when all the groups (triplets) will have been processed, the file will consist of $2N_0$ blocks, in which case we say that a 'full expansion' has taken place. In general, a full expansion may consist of more than two partial expansions. For example, suppose that a full expansion consists of three partial ones. In this case, if N_0 is the initial number of blocks, then $N_0/3$ blocks are created during each partial expansion, therefore, the file finally occupies $2N_0$ main blocks.

The record redistribution after each split in the LHPE scheme is different than the original LH scheme, i.e. redistribution is not performed by considering one more bit of the key suffix. Instead, the home address of each key is calculated in a particular, complicated manner, called 'rejection technique', by using mainly three variables:

- the level of the file,
- the size of the group to split, and
- the value of pointer 'next'.

In addition, the following parameters are involved:

- a hashing function, $h(key)$, to determine the order of the group in the file, and
- a sequence of hashing functions, $G(h_1(key),\ h_2(key),\ \ldots)$, to determine the order of the block in the group.

Further details on this issue can be found in [4]. However, the LHPE scheme has a basic similarity when compared with the LH scheme: The choice of the group of main blocks for which a split is pending, depends linearly on the value of pointer 'next'. In other words, it is irrelevant which main block has overflowed. Hence, the LHPE scheme is an improvement over LH but it does not drastically eliminate the disadvantages of the latter.

The LHPS scheme is based on the LH scheme. In fact, these schemes have identical behavior if there are no overflow records. However, LHPS is more complex because it maintains data about overflowing blocks in two additional structures. The basic idea in LHPS is to split the block with the longest overflow chain rather than wait until this block is pointed at by 'next'. The information concerning the block with the longest overflow chain is extracted from a 'heap structure'. The number of heap nodes equals the number of main blocks of the LHPS file which have overflow chains. The heap is updated when:

- a block overflows for the first time or
- the length of a block's overflow chain is increased or, finally,
- a block split takes place.

It is worth noting that every block is split only once during an expansion. Therefore, the nodes of the heap have to accommodate an additional field, which indicates whether the relevant block is splittable at the present level. Besides the heap structure one more bitmap structure, the 'indicator set', is required to show which block has already split. The length of the indicator set equals the maximum number of blocks of the main LHPS file at the current level. At the beginning (end) of every expansion all the bits of the indicator set are equal to zero (one). It should be noted that

the size of the additional structures represents a small fraction of the size of the actual file and, therefore, they can fit in main memory while the file remains opened. As a consequence, there is no need for additional disk operations and thus time cost metrics are not affected. This organization also maintains a linearly moving pointer which is actually used whenever there are no overflows (the heap is empty) by considering, in addition, the indicator set.

Symbol	Definition
S	: Successful search cost (in block accesses)
I	: Insertion cost (in block accesses)
N_0	: Number of initial main blocks
l	: Number of levels during expansion
len	: Length of a chain of overflow blocks
R_i	: Number of records in the i-th block of a chain
FR	: Number of file records
MB	: Number of main blocks
OB	: Number of overflow blocks
MBC	: Main block capacity
OBC	: Overflow block capacity
OS	: Overflow space (positions per file record)
SUT	: Storage utilization threshold (%)
SUF	: Storage utilization factor (%)

Table 1. Symbol definition list.

Figure 1 shows a comparative example of these organizations. At some point in time an insertion is performed and the file of Figure 1a is produced. This instance is valid for all three organizations described above. By making use of the symbols defined in Table 1, we can make the following observations: (a) the file consists of FR=13 records residing in MB=4 main blocks and OB=3 overflow blocks, (b) the main (overflow) block capacity is MBC=3 records (OBC=1 record), and (c) the mean cost for a successful search equals S=16/13=1.23 block accesses. As a result, the overflow space needed is OS=3/13=0.23 positions per record. Assume that the storage utilization threshold is SUT=85%. From Figure 1a we derive that the storage utilization factor equals SUF=13/15=86.7%. Hence, a split has to take place.

In the LH scheme the first block (address 00) must split. Figure 1b shows the result of this expansion. Now, the new storage utilization factor is SUF=13/18=72.2%, the successful search cost equals S=16/13=1.23 block accesses, therefore the average overflow space requirement is OS=0.23 positions per record. In the LHPE scheme the group which consists of the first and the third blocks (addresses 00 and 10) has to split. Figure 1c shows the result: The new storage utilization factor equals SUF=13/17=76.5%, the average successful search costs S=15/13=1.15 block accesses, and, thus, the average overflow space requirement equals OS=0.15 positions per record. Finally, the expansion in the LHPS scheme is shown in Figure 1d. It can easily be verified that the new cost metrics are the same with those of the LPHE scheme.

3. LINEAR HASHING WITH PARTIAL EXPANSIONS AND PRIORITY SPLITTING

In this section we present a new variation of linear hashing for primary key retrieval. It is a combination of the LHPE and LHPS schemes and we call it *Linear Hashing with Partial Expansions and Priority Splitting (LHPEPS)*. This new organization inherits the merits of both schemes and, in general, it has an improved performance over both of them. The result of its use over the initial file instance, is shown in Figure 1e. In particular, now the storage utilization factor equals SUF=81.3%, the average successful search cost equals S=1.08 block accesses and, thus, the overflow space demands are OS=0.08 positions per record on average.

The LHPEPS scheme makes use of three additional structures:

- An indicator set, which is a bitmap structure. Each bit corresponds to a group of main blocks

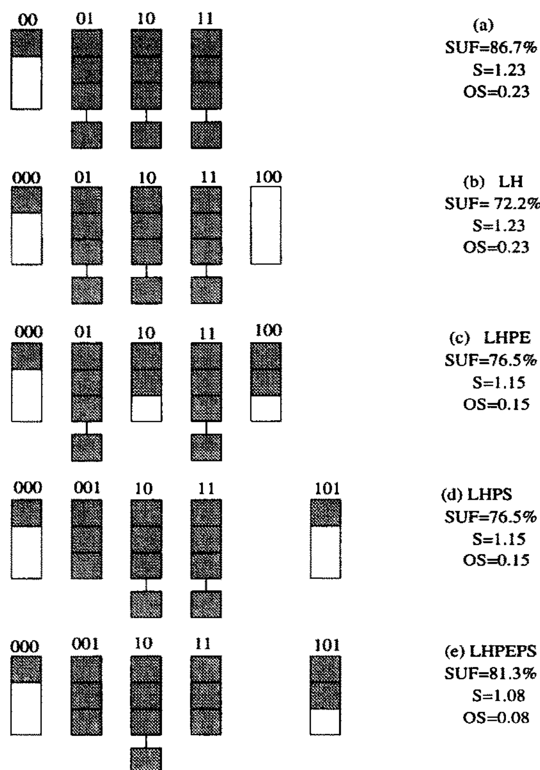

Fig. 1. A file instance and four types of expansions.

which have split during an expansion. It is recalled that in LHPS every bit of the indicator set corresponds to one main block, not to a group of blocks. As in the case of LHPS, at the beginning/end of every expansion the bits of this bitmap are reset.

- A list structure with a number of nodes which equals the number of main blocks which have overflow records. Each node has two fields whose contents declare the length of the overflow block chain and the logical address of the main block, respectively. List nodes are sorted in descending order of the chain length and ascending order of the logical address. This structure is analogous to the heap of the LHPS scheme. and we call it 'list of blocks'.

- A second list structure with a number of nodes, equal to the number of groups of the main blocks which contain overflow records. Each of these nodes has two fields: The contents of the first field represents the total number of overflow blocks per group. The contents of the second one represents the logical address of the group. The nodes of the list are sorted in descending order of the total number of overflow blocks and in ascending order of the logical address. We call this structure 'list of groups'. It has exclusively been designed for the LHPEPS scheme and its necessity stems from the fact that the number of main blocks per group changes in every partial expansion. Therefore, this structure is initialized at the beginning of every partial expansion by scanning the list of blocks which remains unaffected during the expansions. This list is updated whenever a chain is either created or elongated or split.

In total, these structures are relatively small in size, and, therefore, while the file is opened, they can reside in main memory without affecting the time cost metrics. This organization also maintains the pointer 'next' but now it is used only only when there are no overflows.

In our implementation two header nodes are used for the above list structures. The first one points to the blocks or the groups of blocks which have not split during the present cycle and the second one points to the blocks which have split or have been produced during a split. This

implementation is equivalent to having a 'flag field' in every node of the list which indicates whether the relevant block has split or not.

We provide next an algorithm for the manipulation of insertions and expansions in the LHPEPS structure. Some abstraction over the actual implementation is necessary, in order to demonstrate the vital points of the organization. The algorithm has many characteristics in common with the relevant ones of the LHPE and LHPS scheme.

INSERTION AND EXPANSION ALGORITHM

```
Determine the group where the record is directed, by using one of the two
     hashing functions;
Determine the main block among those in the group where the record must be
     inserted, by using the sequence G of hashing functions;
IF this main block is not full THEN insert the record
ELSE {the main block is full}
     IF the record fits in the last block of the overflow chain THEN
          insert the record in this block
     ELSE                { either the record does not fit in the last block }
          BEGIN            { of the overflow chain or no such chain exists }
               create an overflow block; insert the record in this block;
               update the chain; update the list of blocks;
               update the list of groups
          END;
IF the storage utilization factor > the threshold value THEN       { split }
     BEGIN
          IF the list of groups is empty THEN
               get the group shown by pointer 'next';
          ELSE get from the list of groups that one with the largest total
               number of overflows which has not yet split at this level;
          expand the group by creating a new block;
          FOR every record of the group
               IF imposed by the sequence G of hashing functions THEN
                    relocate the record to the new block;
          update the list of blocks; update the list of groups
     END;
IF a group has split THEN
     BEGIN
          update the indicator set;
          update pointer 'next' by using the indicator set
     END;
IF a partial expansion has been completed THEN
     BEGIN
          initialize pointer 'next'; initialize the indicator set;
          increase the group size by one
     END;
IF a full expansion has been completed THEN
     BEGIN
          initialize pointer 'next'; initialize the indicator set;
          initialize the group size; increase the level value by one;
          rebuild the list of groups by using the list of blocks
     END;
```

4. NUMERICAL RESULTS

The original linear hashing and its three variations which were presented in the previous sections have been implemented in Turbo Pascal. An extensive simulation has been carried out in order to compare the four alternative organizations by varying the following design parameters:

- main block capacity (MBC=10,20,50)
- overflow block capacity (OBC=1,3,5,10,20,50,...,MBC), and
- storage utilization threshold (SUT=70,75,80,85,90,95%),

for the estimation of the following performance costs:

- successful search cost,
- insertion cost, and
- overflow space per record.

During the simulation we inserted approximately 20,000 records and estimated the performance measures many times during the insertion phase. The statistics are the mean values of ten experiments.

The time related costs were measured in block accesses. All other overhead time costs (processing of the supporting data structures) were considered to be negligible since the relevant operations are performed in main memory. At each point in time, the average search time cost in block accesses is calculated by using the expression:

$$S = \frac{1}{FR} \sum_{i=0}^{len} R_i \, (i+1) \tag{1}$$

where len is the maximum length of an overflow chain, and R_i is the number of records in the i-th overflow block, (R_0 is the number of records in the main block). The insertion cost (I) equals the number of block accesses (either for reading or for writing) and is measured directly during the simulation.

We consider as space cost only the extra overflow space. This enables us to evaluate how the four methods match the block to split to the block with long overflow chains. More specifically, at each point in time the space cost is measured as the number of overflow positions per file record and are calculated by the expression:

$$OS = \frac{OBC \times OB}{FR} \tag{2}$$

It is apparent that the measures of Figure 1 were produced by using the above formulae.

Every scheme of the linear hashing variations is presented as an improvement over the previous organizations. However, this is not always true because:

- the three performance measures are contradictory to one another, in the sense that gain in one measure may result in extra cost for another,
- given a specific measure, one organization may outperform the others only for some specific ranges of the parameter values.

From our experiments we have concluded that in general the LHPS and LHPEPS schemes are improvements over the LH and LHPE schemes respectively. Therefore, for comparison purposes, the four organizations should be considered as two pairs. These remarks are further explained next.

In Figures 2-5, the average successful search cost is depicted as a function of the total number of file records. For all figures, the parameter values are: MBC = 20 records, OBC = 5 records,

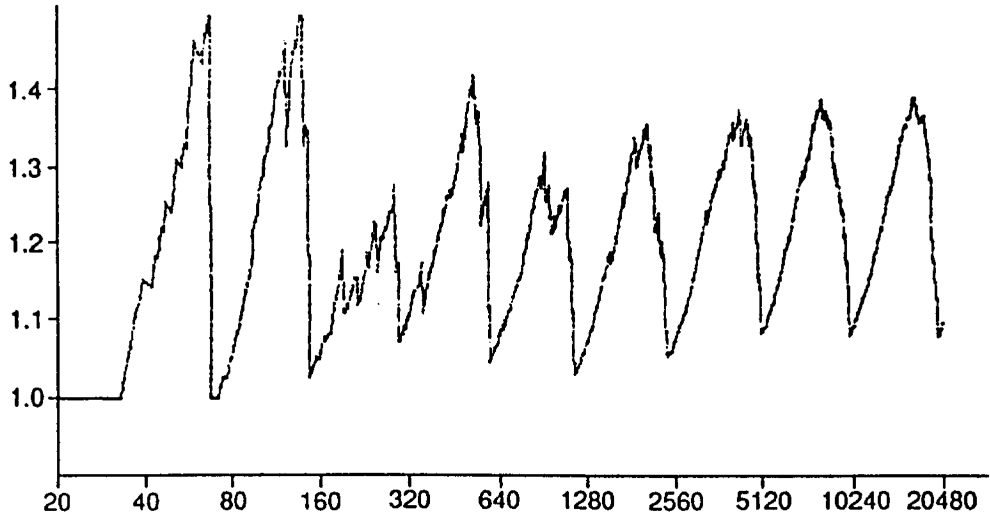

Fig. 2. Mean search cost of the LH scheme as a function of FR.

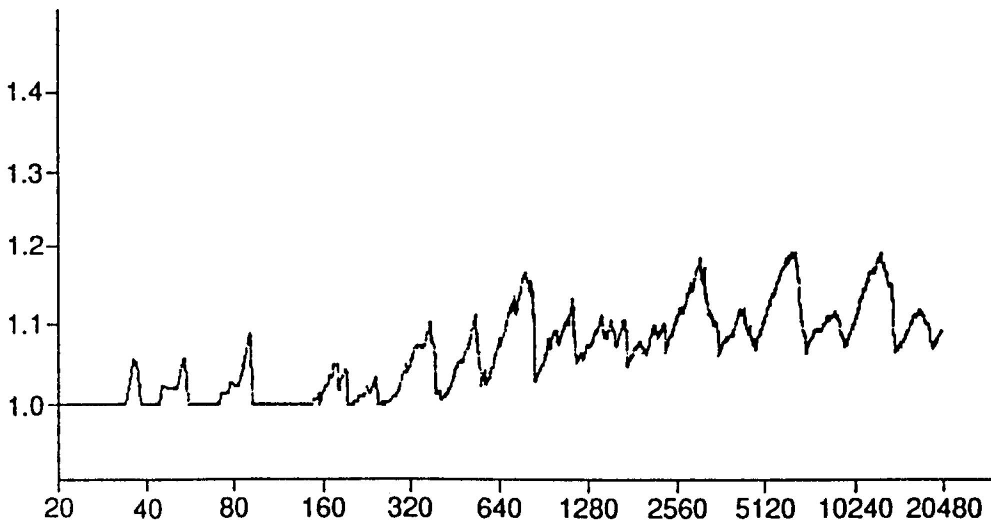

Fig. 3. Mean search cost of the LHPE scheme as a function of FR.

and $SUT = 85\%$. It should be noted that in all figures the search performance is characterized by some sort of periodicity. More specifically, the form of the relevant curves remains the same after the fifth level (2,500 records approximately).

In Figure 2, we notice that the search cost minima of the LH scheme occur at the beginning/end of each level. In contrast, in Figure 3, we can see that there is a local minimum during each level, which occurs at the beginning/end of every partial expansion. In addition, the LHPE performance is better than that of the LH scheme. Figure 4 represents the search cost of the LHPS scheme. The performance improvement over that of the LH scheme is obvious, i.e. the minima and maxima are considerably lower. Figure 5 represents the performance of the LHPEPS scheme. In a similar way, it is obvious that this organization outperforms both the LH and LHPE schemes. Another major advantage of the LHPE scheme, is that it has a very stable behavior. As a result, it has the least search cost variance from all organizations. From our experiments, it has been concluded

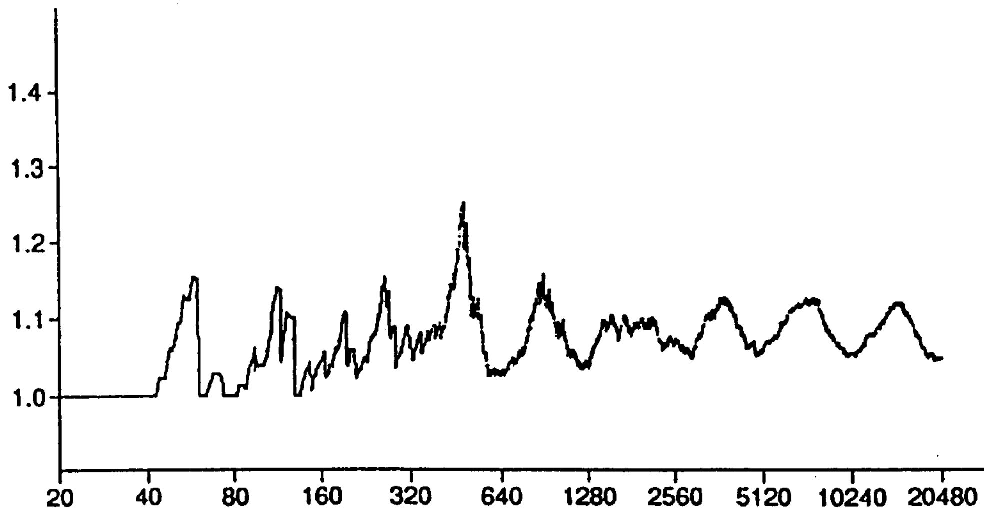

Fig. 4. Mean search cost of the LHPS scheme as a function of FR.

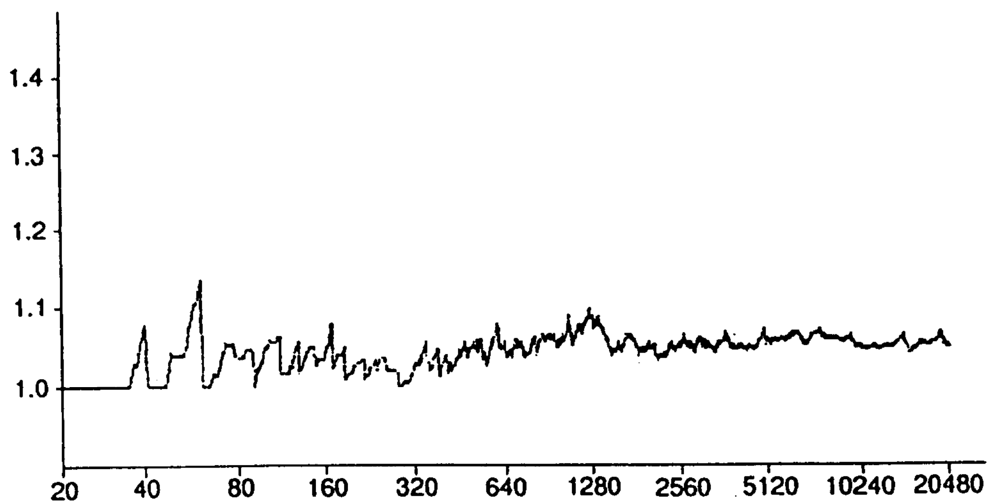

Fig. 5. Mean search cost of the LHPEPS scheme as a function of FR.

that in the case of greater main block capacities and smaller overflow block capacities, the variance of LHPEPS is comparatively much smaller than that of the other organizations.

Another interesting remark of the simulation is the following: Given a main block size and a storage utilization threshold, the time related costs are minimized for a specific value of the overflow block size. With respect to the main block size, this is explained as follows: For a given number of file records,

- small values of overflow block size result in longer overflow block chains and increased response time, whereas
- large values of overflow block size result in low values of the storage utilization factor, the expansion process is delayed and, finally, the time performance deteriorates.

The results of Table 2 concern the LH scheme but the same remarks apply to the other organizations, too. This table shows the search cost as a function of the three parameters (MBC, OBC and SUT). Since the optimum overflow block capacity is practically independent of the threshold, in the remainder we neglect it for purposes of simplicity. Thus, we can see that for main block capacities of 10, 20 and 50 records, the optimum value of the overflow block capacity is 3, 5 and 10 records, respectively. These optimum values have been adopted for all the tables which follow.

		SUT					
MBC	*OBC*	70%	75%	80%	85%	90%	95%
10	1	1.19	1.27	1.38	1.57	1.87	2.43
	3	1.11	1.15	1.21	1.33	1.55	2.39
	5	1.10	1.14	1.22	1.38	1.83	3.73
	10	1.15	1.33	1.57	1.95	2.72	4.90
20	1	1.15	1.27	1.44	1.67	2.03	2.85
	3	1.08	1.14	1.22	1.31	1.49	2.03
	5	1.07	1.10	1.15	1.27	1.46	2.42
	10	1.06	1.10	1.16	1.35	1.98	4.02
	20	1.11	1.31	1.58	1.96	2.74	4.90
50	1	1.23	1.41	1.65	2.22	2.83	4.31
	3	1.08	1.15	1.27	1.43	1.76	2.31
	5	1.07	1.10	1.18	1.31	1.49	1.94
	10	1.04	1.08	1.13	1.20	1.37	2.16
	20	1.04	1.07	1.12	1.21	1.73	3.72
	50	1.09	1.13	1.38	1.69	2.49	4.72

Table 2. Mean search cost of the LH scheme as a function of MBC, OBC and SUT.

		SUT						
MBC	*OBC*	70%	75%	80%	85%	90%	95%	Structure
10	3	1.11	1.15	1.21	1.33	1.55	2.39	LH
		1.05	1.08	1.12	1.19	1.35	2.17	LHPE
		1.01	1.03	1.06	1.12	1.24	2.09	LHPS
		1.02	1.04	1.07	1.13	1.25	2.05	LHPEPS
20	5	1.07	1.10	1.15	1.27	1.46	2.42	LH
		1.02	1.03	1.06	1.11	1.22	2.18	LHPE
		1.01	1.02	1.04	1.09	1.21	2.20	LHPS
		1.00	1.01	1.02	1.06	1.15	2.08	LHPEPS
50	10	1.04	1.08	1.13	1.20	1.37	2.16	LH
		1.00	1.01	1.02	1.05	1.12	1.89	LHPE
		1.01	1.02	1.05	1.10	1.20	2.04	LHPS
		1.00	1.00	1.01	1.02	1.07	1.84	LHPEPS

Table 3. Mean search cost as a function of MBC, OBC and SUT.

We now compare the search performance of the four file organizations. We recall that, in general, every organization outperforms the previous ones, but we should also notice that there are exceptions in this remark. More specifically, we could argue that partial expansions decrease the search cost. Table 3 gives the mean value of the search cost as a function of MBC, OBC and SUT. By examining the effect of the block capacities, we can make some interesting conclusions:

- for small values of the main block capacity (e.g. MBC=10), the LHPS scheme performs better than the others. There is only one exception for large threshold values, where the LHPEPS scheme performs better.

- for large values of the main block capacity (e.g. MBC=50) the LHPEPS scheme is the most preferable structure for any threshold value. In this case, even LHPE is better than LHPS.

By examining the effect of the storage utilization threshold, we can conclude the following:

- for the threshold value of 95%, LHPEPS outperforms any other organization in the improvement over the original LH scheme.
- for the threshold value of 90%, all organizations achieve the greatest improvement over the original LH scheme. More specifically, LHPEPS provides an improvement of approximately 20% over LH, for all MBC, OBC values we have tried.

		SUT						
MBC	*OBC*	70%	75%	80%	85%	90%	95%	Structure
10	3	3.06	3.23	3.49	3.92	4.66	6.92	LH
		3.27	3.39	3.56	3.90	4.57	7.09	LHPE
		2.70	2.80	2.98	3.29	3.86	6.48	LHPS
		3.14	3.21	3.36	3.68	4.25	6.87	LHPEPS
20	5	2.68	2.86	3.10	3.58	4.29	7.00	LH
		2.61	2.72	2.89	3.18	3.74	6.85	LHPE
		2.40	2.49	2.64	2.98	3.56	6.71	LHPS
		2.53	2.59	2.69	2.96	3.51	6.70	LHPEPS
50	10	2.42	2.65	2.91	3.27	3.98	6.49	LH
		2.23	2.28	2.41	2.63	3.13	6.03	LHPE
		2.27	2.37	2.59	2.90	3.44	6.35	LHPS
		2.19	2.22	2.29	2.48	2.90	5.98	LHPEPS

Table 4. Mean insertion cost as a function of MBC, OBC and SUT.

Let us now compare the four organizations with respect to the insertion cost. By examining again the effect of the block capacities in Table 4, we can conclude that:

- for small values of the main block capacity (e.g. MBC=10), the LHPS scheme is far better that any other. We also notice that in many cases (small threshold values) LHPE and LHPEPS have worse performance than that of the original LH scheme.
- For large values of the main block capacity (e.g. MBC=50) LHPEPS is the most preferable structure, since it outperforms any other organization. In the last case, LHPE is better than LHPS.

By examining the effect of the storage utilization threshold, we can conclude the following:

- for the threshold value of 95%, LHPEPS outperforms any other organization in improvement over the original LH scheme.
- once again, for the threshold value of 90%, all organizations yield the greatest improvement over the original LH scheme. More specifically, for this threshold value LHPEPS gives approximately a 25% improvement over the LH scheme.

The reason why these remarks are different than the remarks holding for the search cost, is that during an insertion, both LHPE and LHPEPS process whole groups of blocks whereas, in contrast, the other two organizations process individual blocks. This implies that the partial expansions approach increases the insertion cost.

Now the four file organizations are compared with respect to the required overflow space per file record. By examining the effect of the storage utilization threshold in Table 5, the following can be deduced:

MBC	OBC	SUT 70%	75%	80%	85%	90%	95%	Structure
10	3	0.11	0.14	0.18	0.25	0.34	0.54	LH
		0.07	0.10	0.13	0.19	0.28	0.52	LHPE
		0.02	0.05	0.09	0.15	0.24	0.52	LHPS
		0.04	0.06	0.09	0.16	0.24	0.51	LHPEPS
20	5	0.08	0.11	0.14	0.21	0.30	0.52	LH
		0.03	0.05	0.09	0.13	0.22	0.51	LHPE
		0.02	0.04	0.07	0.12	0.21	0.51	LHPS
		0.01	0.02	0.05	0.10	0.19	0.50	LHPEPS
50	10	0.05	0.08	0.12	0.16	0.24	0.45	LH
		0.01	0.02	0.04	0.07	0.15	0.43	LHPE
		0.02	0.04	0.07	0.11	0.18	0.44	LHPS
		0.00	0.00	0.02	0.05	0.11	0.43	LHPEPS

Table 5. Mean overflow space as a function of MBC, OBC and SUT.

- the smaller the threshold value is, the smaller the required overflow space per file record is.
- the smaller the threshold value is, the greater the relative gain of using the variations over the original organization is. In general, if the threshold value is 95% then the improvement over the original organization is negligible, whereas if the threshold value is 70% then the relative gain is impressive. Notice, however, that the absolute value of the gain is maximized when the threshold value is 90%.

The examination of the effect of the block capacities, yields the following results:

- for small block capacities (e.g. MBC=10) the LHPS scheme has the best performance. There is only one exception: For a threshold value of 95% the LHPEPS scheme is the best.
- for large block capacities (e.g. MBC=50) LHPEPS shows the greatest improvement over the original LH scheme. It is worth noting, in particular, that if the threshold value is 70%, then the gain due to LHPEPS is impressive (nearly 98%).

In order to make conclusions about the space performance, we can equivalently, use the search cost in place of the required overflow space. This is due to the fact that space and search costs are not contradictory in nature. In other words, it is implied that more dense packing results in faster search and vice versa.

5. CONCLUSIONS

Linear hashing for primary key retrieval is considered to be one of the the best dynamic schemes since it does not use an index and has a very good time performance. This organization and two of its variations, *linear hashing with partial expansions* and *linear hashing with priority splitting*, have been presented in this paper. In addition, a new variation of linear hashing has been presented, the **Linear Hashing with Partial Expansions and Priority Splitting (LHPEPS)**.

This organization is a generalization of all the previous approaches and inherits all their merits. Insertion and expansion pseudo-algorithms of this organizations have been given. Deletion is performed in an analogous manner, therefore the relevant algorithm has been omitted for brevity reasons.

We have compared by simulation all four organizations with respect to search and insertion costs and with respect to overflow space. Tables 6 and 7, which summarize the remarks of the previous sections, concern the search and insertion costs. The results for the overflow space have been omitted, since they are almost identical to those of Table 6. By 'small', 'medium' and 'large'

MBC, OBC	*SUT* small	medium	large
small	LHPS LHPEPS	LHPS LHPEPS	LHPEPS LHPS
medium	LHPEPS LHPS	LHPEPS LHPS	LHPEPS LHPE
large	LHPEPS LHPE	LHPEPS LHPE	LHPEPS LHPE

Table 6. Classification with respect to the search cost.

MBC, OBC	*SUT* small	medium	large
small	LHPS LHPEPS	LHPS LHPEPS	LHPS LHPEPS
medium	LHPS LHPEPS	LHPS LHPEPS	LHPEPS LHPS
large	LHPEPS LHPE	LHPEPS LHPE	LHPEPS LHPE

Table 7. Classification with respect to the insertion cost.

block capacities, we mean the pairs (10,3), (20,5) and (50,10) respectively, where the first (second) component indicates the *MBC* (*OBC*). By 'small', 'medium' and 'large' threshold values, we mean the pairs (70%,75%), (80%,85%) and (90%,95%) respectively. Every entry in each of these tables contains the first and second choice for the particular capacity and threshold value.

These tables illustrate clearly the enhanced performance of LHPEPS over its counterparts. We notice that in most of the cases, it has the best performance. In some others it is second but its performance is also close to the best. The tables in the previous section show that the performance of all four techniques deteriorates for a storage utilization threshold equal to 95%. LHPEPS has the best performance, if this threshold value equals 90%, independently of the cost metric. In addition, the accompanying structures of LHPEPS are small and can thus reside in main memory.

Future research includes the integration of the technique of priority splitting in multidimensional linear hashing and recursive linear hashing. More specifically for the first case, it is expected that the number of recursive overflow files will be reduced, since the technique of priority splitting reduces the length of overflow block chains. Another direction involves the integration of our new scheme to accommodate variable length records, along the lines of the technique presented in [11].

Acknowledgements — Thanks are due to Mr. G. Dedeoglou and Mr. G. Papadimos for their valuable help during the experimentation. We are also grateful to the referees, whose comments greatly improved the presentation of this paper.

REFERENCES

[1] R. J. Embody and H.C. Du. Dynamic Hashing. *ACM Computing Surveys*, **20**(2), pp. 85–113 (1988).

[2] R. Fagin ,J. Nievergelt ,N. Pippenger and H. R. Strong. Extendible Hashing - a Fast Access Method for Dynamic Files, *ACM Transactions on Database Systems*, **4**(3), pp. 315–344 (1979).

[3] P.Å Larson. Dynamic Hashing, *BIT*,**18** (2), pp. 184-201 (1978).

[4] P.Å Larson. Linear Hashing with Partial Expansions. *Proceedings of the 6th International Conference on Very Large Data Bases*, pp. 224–232 (1980).

[5] P.Å Larson. Performance Analysis of Linear Hashing with Partial Expansions. *ACM Transactions on Database Systems*, **7** (4), pp. 566–587 (1982).

[6] P.Å Larson. A Single-file Version of Linear Hashing with Partial Expansions. *Proceedings of the 8th International Conference on Very Large Data Bases*, pp. 300–309 (1982).

[7] P.Å Larson. Performance Analysis of a Single-file Version of Linear Hashing. *The Computer Journal*, **28** (3), pp. 319–329 (1985).

[8] P.Å Larson P.Å. Linear Hashing with Overflow Handling by Linear Probing. *ACM Transactions on Database Systems*, **10**,(1), pp. 75–89 (1985).

[9] W. Litwin . Virtual Hashing - a Dynamically Changing Hashing,.*Proceedings of the 4th International Conference on Very Large Data Bases*, pp. 517–523 (1978).

[10] W. Litwin. Linear Hashing - a New Tool for File and Table Addressing. *Proceedings of the 6th International Conference on Very Large Data Bases*, pp. 212–223 (1980).

[11] Y. Manolopoulos and N. Fistas . Algorithms for a Hash Based File with Variable Length Records. *Information Sciences*, **63**, pp. 229–243 (1992).

[12] G.N.N. Martin. Spiral Storage - Incrementally Augmentable Hash Addressed Storage. *Theory of Computation Report #27, University of Warwick, England* (1979).

[13] J.K. Mullin . Tightly Controlled Linear Hashing without Separate Overflow Storage. *BIT*, **21**, pp. 390–400 (1981).

[14] K. Ramamohanarao and R. Sacks-Davis. Recursive Linear Hashing. *ACM Transactions on Database Systems*, **9** (3), pp. 369–391 (1984).

[15] W. Ruchte and A. Tharp. Linear Hashing with Priority Splitting. *Proceedings of the 3th International IEEE Conference on Data Engineering*, pp. 2–9 (1987).

[16] A. Tharp. *File Organization and Processing*, Wiley (1988).