Multidimensional Order Preserving Linear Hashing with Partial Expansions

Hans-Peter Kriegel
Lehrstuhl fuer Informatik I
Universitaet Wuerzburg
D-8700 Wuerzburg
West Germany

Bernhard Seeger
Institut fuer Informatik II
Universitaet Karlsruhe
D-7500 Karlsruhe 1
West Germany

Abstract

We present a new multidimensional dynamic hashing scheme without directory intended for files which grow and shrink dynamically. For previous schemes, the retrieval performance either suffers from a superlinearly growing directory or from an uneven distribution of the records over the pages of a file even for uniform record distribution. Our scheme is a multidimensional order preserving extension of (one-dimensional) linear hashing with partial expansions and thus overcomes these disadvantages. For uniform distribution our scheme performs better than its competitors which is underligned by experimental runs with an implementation of our scheme. In the last section we give a brief outline of the quantile method which guarantees that our scheme performs for a non-uniform distribution practically as well as for a uniform distribution. In addition to its excellent performance, our scheme fulfills all the necessary requirements to be used in an engineering database system: it is dynamic, is suitable for secondary storage devices, supports point data and spatial data objects and supports spatial clustering (proximity queries).

1. Introduction

Concerning database systems for standard applications, e.g. commercial applications, there is a large variety of index structures at the disposal of the database designer for the implementation of the physical level of a database system. As demonstrated in [Kri 84] there are efficient tree-based index stuctures such as multidimensional B-trees, in particular kB-trees.

For non-standard databases, also called engineering databases, used in applications such as image processing, design and manufacturing (CAD/CAM) etc. none of the tree-based index structures is suitable, since they cluster records according to the lexicographical ordering. Hash-based index structures cluster records which are close together in the key space. This is the type of clustering which we need in engineering databases.

However, multidimensional hashing schemes suffer from two shortcomings. In most engineering applications, objects are nonuniformly distributed in space. First, there is no multidimensional hashing scheme which exhibits practically the same retrieval performance for nonuniform distribution as for uniform distribution. Second, even for uniform distribution, the retrieval performance of

all known multidimensional hashing schemes suffers from two shortcomings which will be elaborated in the following paragraph.

Thus we will proceed in two steps. In this paper we will design a multidimensional dynamic hashing scheme which will perform better than its competitors for uniform distribution. For nonuniform distribution all hashing schemes degenerate performancewise. In the last section we will outline a method which applied to our scheme guarantees for nonuniform distributions practically the same performance as for uniform distribution. None of the previous schemes for multidimensional data achieves this performance. In this paper we will primarily focus on uniform distributions, which is the explicit or implicit assumption in all publications dealing with multidimensional hashing.

Multidimensional hashing schemes fall into one of two broad categories: those that do not use a directory and those that use a directory. Typical representatives of the second category are the grid file [NHS 84] and multidimensional extendible hashing (MEH) [Oto 84]. Unfortunately, all methods with directory which a) guarantee two disk accesses for an exact match query and b) are dynamic (in the sense that there is no upper bound for the number of records) exhibit a superlinear growth of the directory in the number of records, even when the records are uniformly distributed in the key space. In case of nonuniform distributions the directory may require exponential space.

Since last year a series of variants either of the grid file or of multidimensional extendible hashing have been suggested all distributing the directory on several levels and accomodating as much as possible of the directory in central memory: the 2-level grid file [Hin 85], the multilevel grid file [KW 85], the balanced multidimensional extendible hash tree [Oto 86] and the interpolation-based grid file [Ouk 85]. As indicated by the name, in [Ouk 85] the concepts of the grid file and interpolation-based index maintenance [Bur 83] are integrated. As shown in the paper [Ouk 85] the interpolation-based grid file grows *linear in the number of data pages* . However, as not even mentioned in this paper, the directory may grow *exponential in the number of records* , since a split of a directory page may propagate recursively. In [KW 85] and [Oto 86] the directory is partitioned using the K-D-B-tree concept [Rob 81]. However, for large files, the two-disk-access-guarantee for exact match queries does not hold any more. Depending on the file sizes three or four disk accesses may be necessary to answer an exact match query. Summarizing we can say that none of the structures with directory and no overflow records guarantees linear growth of the directory in the number of records and guarantees two disk accesses for an exact match query at the same time.

For most applications a linearly growing directory is a must. This was the basic motivation for the Symmetric Dynamic Index Maintenance (SDIM) scheme suggested by Otoo [Oto 85]. In contrast to the grid file and MEH, SDIM uses a control function that allows overflow records and guarantees the linear growth of the directory. The SDIM organization is comprised of two levels: a first level of a directory, and a second level of data pages which hold the records. Using the storage mapping function G_{SDIM} each record is mapped onto a directory entry containing a pointer to the header page of a short chain of data pages. By its design, SDIM avoids the performance penalty due to a superlinearly growing directory. However, there still remains a major performance penalty in SDIM which is not mentioned in [Oto 85]. Even in the case of a uniform distribution, SDIM may produce chains with relative load factor 1, 1/2,.., $1/2^d$, where d is the number of attributes. Obviously, this large variance of the load factors leads to poor performance of retrieval and update algorithms.

Another very important drawback of a directory based hashing scheme is its undynamic behavior in case of insertions and deletions. If an insert operation requires introducing a new partitioning hyperplane, this will necessitate $O(n^{1 - 1/d})$ accesses to secondary storage.

Our goal is the design of an efficient multidimensional dynamic hashing scheme which uses no directory and creates chains with relative load factors distributed between 0.5 and 1. Using no directory we have to allow overflow records which are organized by bucket chaining. We consider a page to be a chain of blocks, where the first block, the primary block, is put in a primary file and the other blocks, the secondary blocks, are put in a secondary file.

Looking at the development of onedimensional hashing schemes, we see that the problem of largely varying relative load factors already deteriorates performance of linear hashing [Lit 80]. Larson suggested a very efficient solution of the problem: linear hashing with partial expansions (LHPE) [Lar 80]. The major difference compared with linear hashing is that doubling the file size is done in p partial expansions, where p=2 turned out to be the best compromise. The expected number of page accesses for a successful search is very close to 1. Thus our goal is to suggest a multidimensional order preserving extension of linear hashing with partial expansions.

In the following, we consider a file of d-attribute composite keys $K = (K_1,..,K_d)$. We assume the domain of the keys to be the unit d-dimensional cube $[0, 1)^d$. Obviously, this requirement can easily fulfilled for an arbitrary domain by simple transformation. Our scheme should support the following operations efficiently: insertion and deletion, exact match query, range query and other proximity queries.

This paper is organized as follows. In section 2 we give a brief description of linear hashing with partial expansions and suggest an order preserving variant. In the third section we present our scheme, multidimensional order preserving linear hashing with partial expansions (MOLHPE). In particular, we introduce the address function of our scheme and show how partial expansions are performed. A precise description of the algorithms for exact match and range query in a modula 2 - like formulation follows in section 4. In the fifth section, we report on the performance of an implementation of our scheme. In section 6 we give a short outline of the quantile method which guarantees for non-uniform distributions practically the same ideal behavior of our scheme as for uniform distributions. Section 7 concludes this paper.

2. Linear Hashing with partial expansions

In this section we are dealing with hashing schemes for one dimensional keys. All multidimensional schemes are a generalization of a onedimensional scheme.

The original linear hashing scheme proposed by Litwin [Lit 80] considers a file of N pages with addresses 0,1,..,N-1 at the beginning . When the page j is split, the file will be extended by one page with adress N+j, j = 0,..,N-1. The predetermind sequence of splitting is given by 0,1,..,N-1, i.e. first page 0 is split into page 0 and page N, then page 1 is split, etc.. A pointer p keeps track of the page, which will be split next. If all N pages are split, the file size has doubled, the pointer p is reset to 0 and the splitting process starts over again. After doubling the file size a full expansion is finished.

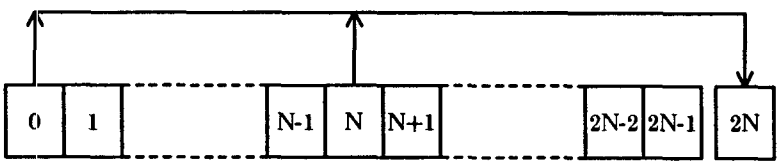
There are two important remarks for constructing an access algorithm. First the decision, whether a record remains in the old page or moves to the new page depends only on the key of the record. Second, the old and the new page should be filled uniformly, i.e. half of the records should remain in the old page. It follows from the second remark that the record load over the file is non - uniform, because the load of split pages can only be half of the load of non - split pages. To achieve a more even load distribution we must partition the pages into groups. If a new page is added to the file, one group of pages will be expanded by one page. Let us consider a group size of

two pages at the beginning. If every group is expanded by one page, we will say the first partial expansion is finished, i.e. the file size increases to 1.5 times of the orginal size. After expanding all groups one more time the file size has doubled and a full expansion has been finished.

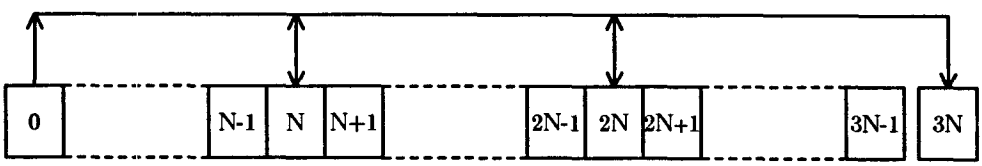
Assuming two partial expansions per full expansion, we start from a file of 2N pages divided into N groups of two pages. Group j consists of pages (j, j + N), j = 0,1,..,N-1. The file will be expanded by group 0, group 1, etc. until the first expansion will be finished. If we expand the j-th group, one third of the records from page j and from page j+N should be relocated in the new page 2N+j. Then we consider N groups of three pages (j, j+N, j+2N), j = 0,..,N-1, and we start the second partial expansion. During the second partial expansion new pages will be created with addresses 3N+j, j = 0,..,N-1.

Example 2.1

(i) Splitting during the first partial expansion (p = 0)



(ii) Splitting during the second partial expansion (p = 0)



The choice of the hashing function is very important for the sequential access to the records. We propose an order preserving hashing function, i.e we can sort the pages p_i, i = 0,..,M in such a way that the keys of all records in page p_{i-1} are smaller than the keys of records in page p_i, i = 1,..,M.

In particular, we choose the following hashing function

$$h(K, L, p) = \begin{cases} \sum_{j=0}^{L-1} b_{j+1} 2^j \text{ , if } \sum_{j=0}^{L-1} b_{j+1} 2^j < 2^{L-1} + p \\ \sum_{j=0}^{L-2} b_{j+1} 2^j \text{ otherwise} \end{cases}$$

where key K $\in$ [0, 1) has the binary representation $\sum_{j \geq 1} b_j 2^{-j}$.

In the above hashing function the level L of the file indicates, how often the file size has doubled and p points to the first page in the group of pages to be expanded next. The following algorithm which is applicable only for two partial expansions is similar to other proposals.

2.2 Algorithm Hash(K, L, p, ex)

```
(* K = Σ_{j≥1} b_j 2^{-j} *)
BEGIN
    group:= Σ_{j=1}^{L-2} b_{j+1} 2^j;        edge:= Σ_{j=1}^{L-2} b_j 2^{-j};
    IF group ≥ p THEN
        q:=ex + 1
    ELSE
        q:=ex + 2
    END;
    (2b_1 + b_0 ):= TRUNC(q * 2^{L-1} * ( K - edge ) );
    (* b_0, b_1 ∈ { 0,1 }  *)
    IF q = 2 THEN
        buddy := b_0
    ELSE
        buddy := 2*b_0 + b_1
    END;
    RETURN ( 2^{L-1} * buddy + group)
END Hash.
```

Algorithm 2.2 uses the following variables: K is the key, L is the level of the file, p points to the first page in the group of pages to be expanded next, ex $\in \{1,2\}$ indicates which of the two partial expansion is presently performed, group gives the group of pages and buddy the page of that group, in which the record with key K resides, and edge gives the left edge of the interval corresponding to the group.

We will generalize order preserving linear hashing with partial expansions to the multidimensional case in the next section. For further considerations of partial expansions the reader is referred to [Lar 80] and [RL 82] and for order preserving hashing schemes we refer to [Ore 83].

3. The address function

We want to present an address function G which is well known from multidimensional extendible hashing (MEH) [Oto 84]. MEH is a directory based hashing scheme, in which the directory is organized as a multidimensional dynamic array addressed by the function G. In the components of the array there are pointers referencing the data pages. In our approach, we will use the function G to address pages, more exactly chains of blocks (since we allow overflow records) without a directory.

The address function G of level L computes addresses $0,..,2^L-1$, where each dimension is treated equally. Each dimension j has an own level L_j, $j = 0,..,d$, which is given by

(3.1) $\quad L_j = L \text{ DIV } d + 1 \qquad j \in \{1,..,L \text{ MOD } d\}$

$\qquad\quad L_j = L \text{ DIV } d \qquad\quad L \text{ MOD } d < j \leq d$

First we evaluate by a special application of algorithm (2.2) an index array $I = (i_1,..,i_d)$.

(3.2) $i_j = \text{Hash}(K_j, L_j, 0, 1) \qquad j = 1,..,d$

The function G is given by

(3.3)

$$G(i_1,..,i_d) = \begin{cases} i_z * \prod_{j \in M} J_i + \sum_{j \in M} c_j * i_j & \text{, if } \max\{i_1,..,i_d\} \neq 0 \\ 0 & \textit{otherwise} \end{cases}$$

where z, M, t, J_i, c_i are given as follow:

$z = \max \{ j \in \{1,..,d\} \mid \text{TRUNC}(\log_2 i_j) = \max \{\text{TRUNC}(\log_2 i_k) \mid k \in \{1,..,d\} \} \}$

$M = \{ 1,..,d \} \setminus \{ z \}$

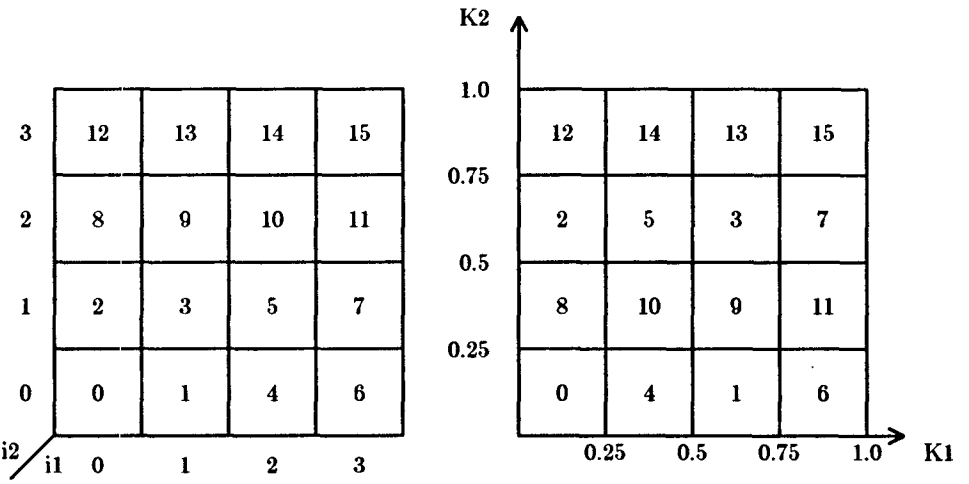
$t = \text{TRUNC}(\log_2 i_z)$

$$J_i = \begin{cases} 2^{t+1} & \textit{, if } i < z \\ 2^t & \textit{otherwise} \end{cases} \qquad , i \in M$$

$$c_i = \prod_{\substack{r=j+1 \\ r \neq z}}^{d} J_i \quad , i \in M$$

Example 3.4

File of level L = 4 addressed by G (d = 2).



On the left side of the illustration we depict the addresses produced by the function G dependent on the Index I = (i_1, i_2). On the right side we illustrate the addresses dependent on the domain of the keys. Every address corresponds to a rectangle, for example address 5 corresponds to rectangle [0.25, 0.5) × [0.5, 0.75).

Our goal is to double the file size not in one step, but to expand the file page by page using an expansion pointer ep = (ep_1,..,ep_d). Similar as in one dimensional LHPE, we consider groups of pages, which will be expanded by one page in order to improve retrieval performance. Up to now we have considered two partial expansions. All the following algorithms are given for this case. Nevertheless we can generalize the algorithms to an arbitrary number of partial expansions. However, two partial expansions turned out to be a good compromise between the cost for inserting and searching a record [Lar 80].

Let us consider a file of size 2^L pages, L $\geq$ d. We will designate the dimension s = L MOD d + 1 as the dimension, in which the expansion will be carried out. During the first partial expansion, we consider groups of two pages addressed by

$$(G(i_1,..,i_s,..,i_d), G(i_1,..,i_s + 2^{L_s - 1},..,i_d))$$

$$0 \leq i_j < 2^{L_j} \quad j \in \{1,..,d\} \setminus \{ s \}$$
$$0 \leq i_s < 2^{L_s - 1}$$

This group of pages will be expanded by one page with address

$$G(i_1,..,i_s + 2^{L_s},..,i_d)$$

One third of the records of the old pages should be relocated in the new page.

The expansion pointer ep = (ep_1,..,ep_d) refers to the group of pages, which will be expanded next. The address of the first page in the group is given by G(ep). At the beginning of a partial expansion the pointer is initialized to

ep := (0,..,0)

If a group is expanded during a partial expansion, the pointer ep will be updated by the algorithm NextPointer.

3.5 Algorithm NextPointer (ep, L)

```
BEGIN
    IF s ≠ 1 THEN i:=1 ELSE i:=2 END;
    LOOP
        ep_i := ep_i + 1;
        IF ep_i = 2^{L_i} THEN
            p_i := 0;
            i:=Next(i)
            (* IF i = k THEN i:=s END *)
            (* IF i = s - 1 THEN i:=s+1 ELSE i:=i+1 END *)
        ELSE
            EXIT
        END
    END (* LOOP *)
END NextPointer.
```

After the first partial expansion has been completed, the file size has increased by the factor 1.5. Then the second partial expansion is started considering groups of three pages addressed by

$$(G(i_1,..,i_s,..,i_d), G(i_1,..,i_s + 2^{L_s - 1},..,i_d), G(i_1,..,i_s + 2^{L_s},..,i_d))$$

This group of pages will be expanded by one page with the address

$G(i_1,..,i_s + 2^{L_s-1} + 2^{L_s},..,i_d)$

$0 \leq i_j < 2^{L_j} \qquad j \in \{1,..,d\} \setminus \{ s \}$

$0 \leq i_s < 2^{L_s-1}$

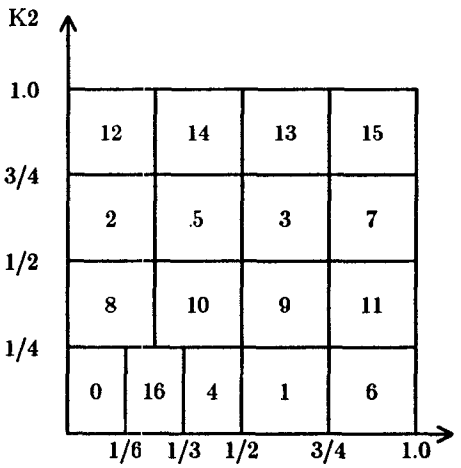
The splitting process of the second partial expansion proceeds in a way analgous to the first partial expansion and therefore will not described here. The following example 3.6 demonstrates the growth of the file during both partial expansions. The addresses of the pages depend on the domain of the keys.

Example 3.6

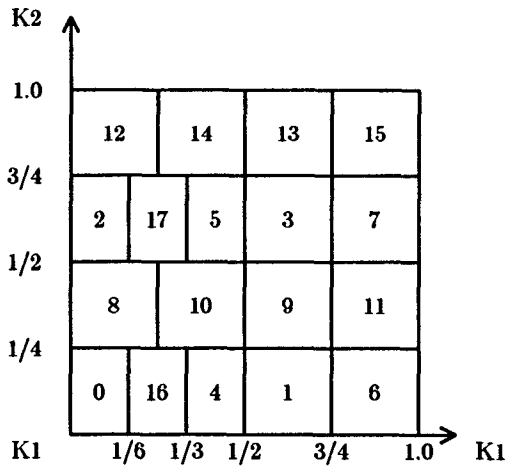
(i) Let us consider a file of level L = 4 (see example 3.2). At the beginning of the first partial expansion, the pages are divided into groups as follows :

(0, 4), (1, 6), (8, 10), (9, 11), (2, 5), (3, 7), (12, 14), (13, 15)

(ii) after the 1st expansion of the group G(ep), ep = (0, 0)

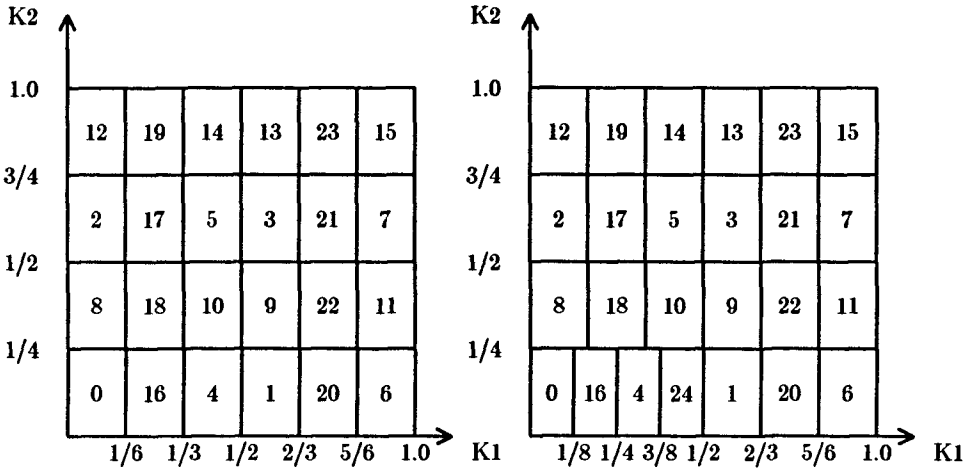


(iii) after the 2nd expansion of of the group G(ep), ep = (0, 1)



(iv) after the 1st partial expansion has been completely finished

(v) one more expansion during the second partial expansion of the group G(ep), ep = (0, 0)



4. Queries

Let us first present the address computation algorithm for multidimensional order preserving linear hashing with partial expansions (MOLHPE). It can be used to answer exact match queries. This algorithm is based on the algorithm Hash (2.2) and additionally incorporates the address function G proposed in the last section. Let us emphasize that the retrieval algorithms presented in this section assume two partial expansions per full expansion.

4.1 Algorithm MultiHash (K, L, ep, ex)

(* $K = (K_1,..,K_d)$ mit $K_j = \sum_{i \geq 1} b_i^j 2^{-i}$ *)

BEGIN

group[j] := $\sum_{i=0}^{L_s - 1} b_{i+1}^j 2^i$ $j \in \{1,..,d\} \setminus \{ s \}$;

group[s] := $\sum_{i=0}^{L_s - 2} b_{i+1}^s 2^i$;

edge := $\sum_{i=0}^{L_s - 2} b_i^s 2^{-i}$;

IF (group[s], group) $\geq^L$ (ep[s], ep) THEN

(* group of pages isn't expand *)

q := ex + 1

ELSE

q := ex + 2

END;

$(2 * b_1 + b_0)$:= TRUNC ($q * 2^{L_s - 1} * (K_s$ - edge));

```
    IF q = 2 THEN
        buddy := b_0
    ELSE
        buddy := 2 * b_0 + b_1
    END;
    RETURN G ( group[1],..,group[s] + buddy * 2^(L_s-1),..,group[d] )
END MultiHash.
```

Algorithm 4.1 uses the following variables : K is the multidimensional key, L is the level of the file, ep is the expansion pointer, ex $\in \{1,2\}$ indicates which of the two partial expansions is presently performed, group gives the address G(group) of the group of pages in which the wanted record resides and buddy determines the page within the group. The operator $\geq$ denotes of lexicographical ordering.

Let us remark that in the case d = 1 the algorithm MultiHash is identical to the algorithm Hash. Furthermore the description complexity of the algorithm is practically not increased by the concept of partial expansions.

Example 4.2

Let us consider the situation of example 3.4 (iii). The invocation of the procedure

MultiHash((0.25, 0.66), 4, (0, 2), 1)

results in the following situation :

group = 0, edge = 0, q = ex + 2 = 3, buddy = 2

G (4, 1) = 17, RETURN(17)

Now let us consider range queries where we specify a pair of bounds, Klow, Khigh $\in [0, 1)^d$ such that $Klow_j \leq Khigh_j$, $j \in \{1,..,d\}$, is fulfilled. Then we ask for all the records with key K in the d - dimensional rectangle $R(Klow, Khigh) = \bigtimes_{j=1}^{d} [Klow_j, Khigh_j)$. In the following we present the algorithm RangeQuery, which produces a range query.

4.3 Algorithm RangeQuery (Klow, Khigh, L, ep, ex)

1. Evaluate the index grouplow, grouphigh of the group and the left limit of the interval edgelow, edgehigh correspondending to the keys Klow and Khigh.
2. actgroup := grouplow;
   FOR i = 1,2 DO
       q_i := ex + i;
       low_i := TRUNC(q_i * 2^{L_s-1} * ($Klow_s$ - edgelow));
       $high_i$:= TRUNC(q_i * 2^{L_s-1} * ($Khigh_s$ - edgehigh));
   END;
3. REPEAT
       IF (actgroup[s], group) $\geq$ (ep[s], ep) THEN
           j := 1
```
```

```
        ELSE
            j := 2
        END;
        IF grouplow[s] = grouphigh[s] THEN
            from := low_j; to := high_j
        ELSIF actgroup[s] = grouplow[s] THEN
            from := low_j; to := q_j - 1
        ELSIF actgroup[s] = grouphigh[s] THEN
            from := 0; to := high_j
        ELSE
            from := 0; to := q_j - 1
        END;
        FOR i := from TO to DO
            actpage := actgroup;
            ( 2 * b_1 + b_0 ) := i ;        (* b_1, b_0 ∈ { 0, 1} *)
            IF q_j = 2 THEN
                buddy :=b_0
            ELSE
                buddy := 2 * b_0~ + b_1
            END; actpage[s] := actpage[s] + 2^(L_s-1) * buddy;
            adr := G( actpage )
            (* Now we can fetch the page with address adr into core and prove the records,
            wether they lay in the region of the query. *)
            end := actgroup = grouphigh;
        END;
        Update the variable actgroup ;
    UNTIL end;
END RangeQuery.
```

We want to emphasize that the algorithm evaluates only addresses of pages, which intersect the region R(Klow, Khigh) of the range query. Even every group of pages we access only to those pages of the group which intersect the region of the query. This is taken care of in the IF - statement at the beginning of the REPEAT - statement. The variable actgroup indicates the actual group intersecting the region of the query.

Since records which are close together in data space reside most likely in one page (if not then in neighboring pages) we can answer proximity queries, such as near neighbor queries very efficiently. This is as well due to the fact that given a page we can easily compute the adresses of all neighbor pages.

5. Performance of MOLHPE

In order to demonstrate the performance of our scheme, we have implemented MOLHPE for one, two and four partial expansions in Modula 2 on an Olivetti M24 PC under MSDOS. Let us mention that MOLHPE using one partial expansion has the same performance as the interpolation based

scheme [Bur 83]. The results from all our experiments can be summarized in the statement that for uniform distribution MOLHPE has the same performance as LHPE and thus is superior to its competitors. Introducing order preserving hashing functions and extending LHPE to the multidimensional case does not cost any performance penalty.

In our experimental set-up we consider only the case of the growing file, i.e. we insert records into the file, but do not delete them. Furthermore, we will assume that each record will be accessed with equal probability. The parameter which is fixed in all our experiments is the capacity of a primary block containing 31 records. From all our experiments, we will present the following selection:

Experiment 5.1:
This experiment was performed for uniform distibution of 2-dimensional records. The maximum file size was 30000 records corresponding to level L = 10. The following parameters were varied within this experiment: the control function for the expansion of the file and the capacity of a secondary block. Although the number of partial expansion were varied from one to four, we present only the results for two partial expansions in this paper.

5.1.1:
i) expansion after c = 28 insertions
ii) capacity s of a secondary block is s = 7 records
Figures 5.2 and 5.3 depict the average number of successfull and unsuccessfull exact match query during the full expansion of the file from level L = 9 to L = 10, corresponding to a doubling of the file from 15000 to 30000 records.

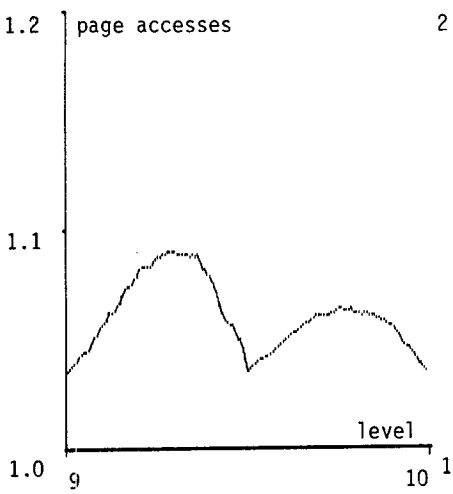


figure 5.2

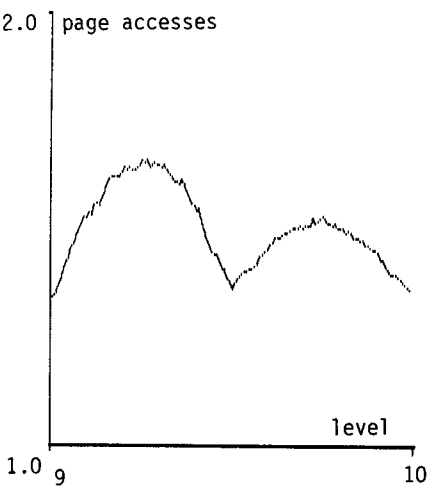


figure 5.3

For the number of page accesses in an exact match query and for the storage utilization, table 5.9 lists the maximum value, the minimum value and the average over a full expansion. Table 5.9 is

shown after presentation of all experiments. Furthermore, the maximum number of blocks in a chain, denoted by maxblock, is listed in this table. If maxblock is not more than two we have a two-disk-access-guarantee for successful and unsuccessful exact match query like for the grid file.

5.1.2:

i) expansion after c = 28 insertions

ii) capacity s of a secondary block is s = 31 records

In this experiment the capacity of the secondary block is increased in order to fulfill the two-disk-access-guarantee (maxblock = 2 in this experiment). Again we depict the average number of page accesses in a successfull and unsuccessfull exact match query during the full expansion of the file in figure 5.4 and 5.5. Comparing table 5.9 it turns out that in this set-up which give the two-disk-access-guarantee at the expense of sacrificing storage utilization (whith an average of 70% will not worse than the grid file) and a slight improvement in exact match performance compared with experiment 5.1.1. Compared with the grid file we have considerably improved average exact match performanace. The improvement in range query performance compared with the grid file will be more drastic.

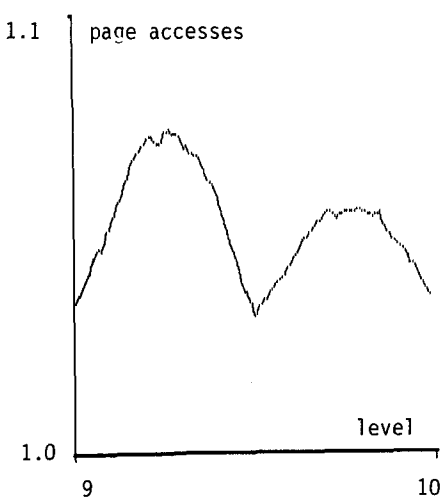


figure 5.4

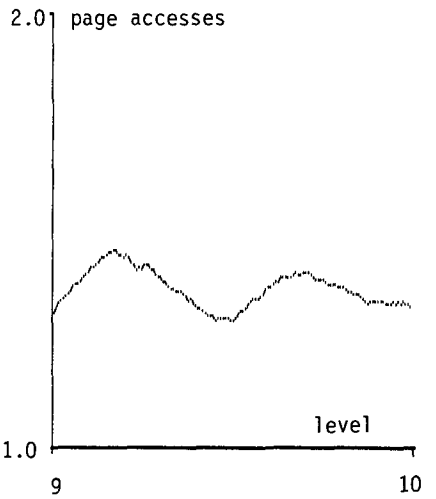


figure 5.5

5.1.3:

i) expansion after c = 21 insertions

ii) capacity s of secondary block is s = 7 records

In this experiment, the expansion is performed already after 21 insertions. This will improve average exact match performance to practically ideal values, in case of successfull exact match queries to 1.005 page accesses, for unsuccessfull ones to 1.065 page accesses. Average storage utilization with 67% is in the grid file range.

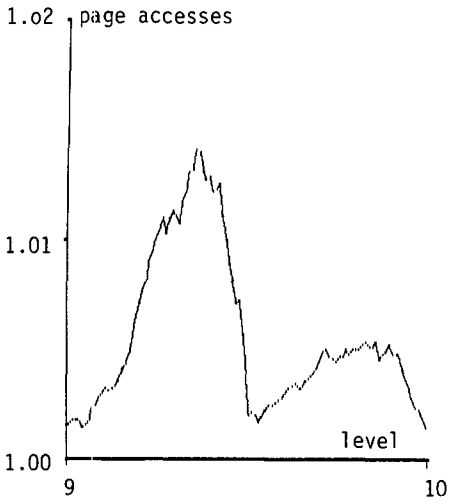


figure 5.6

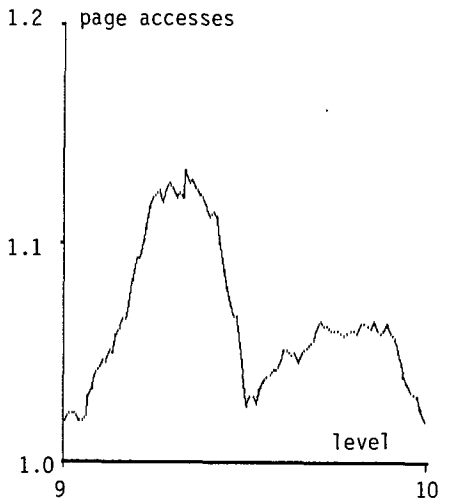


figure 5.7

Experiment 5.8:

This experiment was performed for a non - uniform distribution, where objects are concentrated in the left lower coner (south-west coner) of the unit square $[\ 0,\ 1)^2$. More exactly, for $K_i \in [\ 0,\ 1\ )$, i $= 1,2$, we have:

$Pb(\ K_i < 1/4\ ) = 9/16$

$Pb(\ K_i < 1/2\ ) = 5/16$

where Pb denotes the probability with which K_i is in the specified intervall, i $= 1,2$. Howver we accept component keys K_i which f;;ow the distribution and are not in [0.6, 0.7]. Table 5.9 shows that for such a non - uniform distribution retrieval performance degenerates. Even the maximum number of page accesses in a successfull exact match query of 4.36 is still bad enough. In this respect MOLHPE is not better than any other multidimensional dynamic hashing scheme. Considering the grid file, such a non - uniform distribution will blow up the grid directory, which will result for example in very bad range query performance. Summarizing we can say that MOLHPE turned out to be superior to other schemes for uniform record distributions.

experiment		successful emq	unsuccessful emq	storage utilization	maxblock
5.1.1	maximum	1.090	1.658	0.8498	5
	minimum	1.035	1.340	0.8188	
	average	1.060	1.503	0.8339	
5.1.2	maximum	1.074	1.454	0.7362	2
	minimum	1.031	1.290	0.6677	
	average	1.052	1.366	0.7023	
5.1.3	maximum	1.014	1.132	0.6737	4
	minimum	1.001	1.018	0.6633	
	average	1.006	1.065	0.6693	
5.8	maximum	4.356	9.915	0.6093	26
	minimum	3.728	8.627	0.6019	
	average	3.992	9.193	0.6053	

table 5.9

6. Handling non - uniform record distributions

In the last section, we have seen how the performance of MOLHPE may degenerate for non-uniformly distributed records. In this section we want to give a short outline of the quantile method which guarantees that MOLHPE performs for a non - uniform distribution practically as well as for a uniform distribution.

MOLHPE partitions the key space using equidistant partitioning hyperplanes and thus exhibits optimal performance for uniform record distribution. The more the distribution of records differs from uniform, the worse MOLHPE will perform. We will overcome this disadvantage by selecting the partitioning points depending on the distribution of objects.

Let us demonstrate our method considering a 2-dimensional distribution function F with marginal distribution functions f_1 and f_2. We assume that the stochastic variables K_1 and K_2 are independent, i.e.

$$F(K_1, K_2) = f_1(K_1) * f_2(K_2)$$

For $a \in [0, 1]$ the a-quantile of the 1st (2nd) attribute is the domain value x_a (y_a) such that

$$f_1(x_a) = a \qquad (f_2(y_a) = a)$$

Now let us assume that starting from an empty file we have to partition the key space for the first time and we decide to partition the first dimension (axis). If f_1 is a non-uniform distribution function, we will not partition the first dimension in the middle, but we will choose the 1/2-quantile as the partitioning point. This guarantees that a new record will be stored with equal probability in the page corresponding to the rectangle $[0, x_{0.5}) \times [0, 1)$ or in the page corresponding to the rectangle $[x_{0.5}, 1) \times [0, 1)$, see figure 6.1. During the next expansion we will partition the second dimension (axis) and we will choose the partitioning point $y_{1/2}$, see figure 6.2.

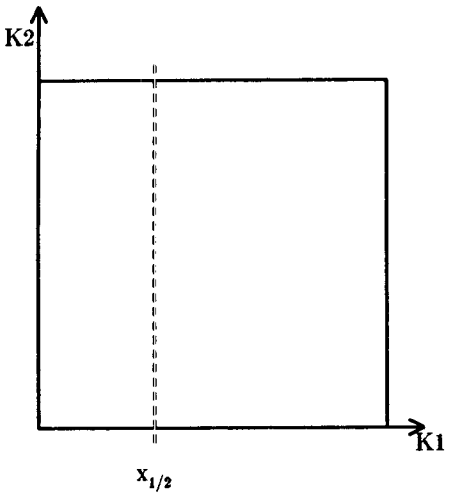


figure 6.1

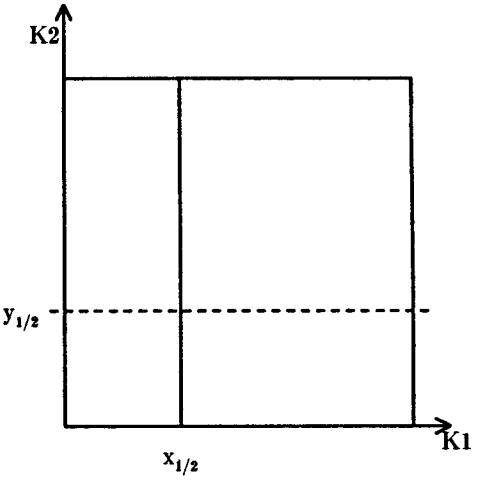


figure 6.2

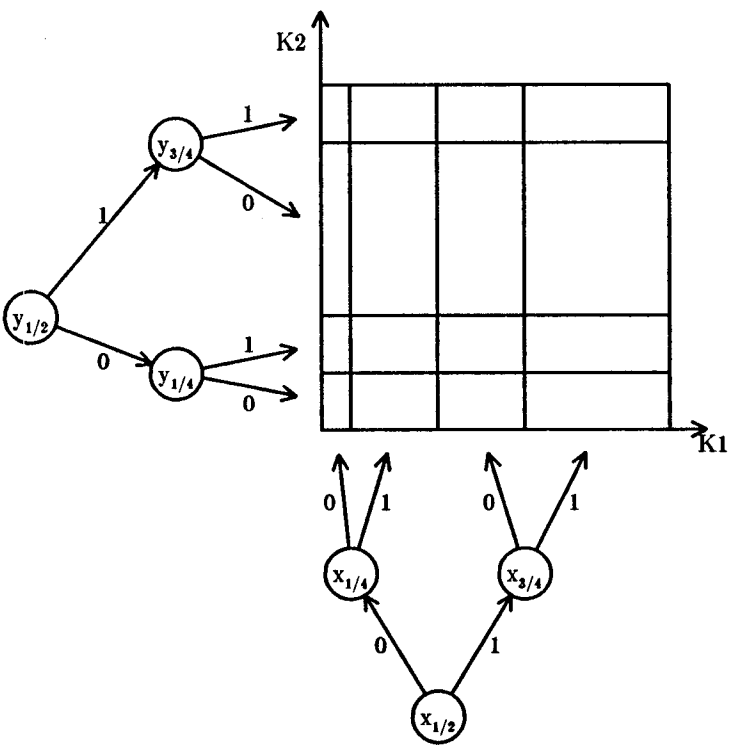


figure 6.3

Figure 6.3 shows the file at level L = 4 and ep = (0, 0), where each axis has been partitioned at the 1/4, 1/2 and 3/4 quantile. As depicted in figure 6.3, the partitioning points are stored in binary trees which can easily be stored in main memory. The *most important property* of these binary trees of partitioning points is the following : for each type of operation a nonuniformly distributed query value K_j, j = 1,2 , is transformed into a uniformly distributed a $\in$ [0, 1) by searching the corresponding binary tree of partitioning points. This uniformly distributed a $\in$ [0, 1) is then used as an input to the retrieval and update algorithms of MOLHPE.

In most practical applications the distribution of objects is not known in advance and therefore we will approximate the partitioning points based on the records presently in the file using a stochastic approximation process. Since we do not know the exact partitioning point (a-quantile) we are dealing with estimates, instead. Now, when the approximation process yields a new estimate we have to shift the corresponding partitioning line and reorganize the file. If this is done in one step, it requires $O(n^{1-1/d})$ page accesses and thus our method would not be dynamic any more. At this point we refer to the paper [KS 86] which deals with all the problems of the quantile method. Concerning the above problem we can assure the reader that reorganization of the file can be performed stepwise requiring only a constant number of page accesses. This is due to the fact that MOLHPE uses no directory. The interested reader is referred to [KS 86]. For the non - interested reader we can summarize this paper in the following statement : the quantile method applied to MOLHPE exhibits for non-uniform record distributions practically the same ideal performance as MOLHPE for uniform distributions. This emphasizes once more the importance of a multidimensional hashing scheme which performs optimally for uniform distributions. For a non-uniform distribution the quantile method can at best exhibit the performance which the underlying scheme has for a uniform distribution.

7. Conclusions

For uniform record distributions we intended to suggest a multidimensional dynamic hashing scheme with no directory which exhibits best possible retrieval performance. The retrieval performance of all known multidimensional hashing schemes either suffers from a superlinearly growing directory or from a very uneven distribution of the records over the pages of the file. By using no directory and applying the concept of linear hashing with partial expansions to the multidimensional case we have overcome both performance penalties. Furthermore, our scheme is order preserving and therefore suitable for engineering database systems. Considering all the ingredients from which our scheme is composed we call it multidimensional order preserving linear hashing with partial expansions (MOLHPE). First experimental runs of an implementation of MOLHPE show that our scheme performs better than its competitors for uniform distribution.

However, as its competitors MOLHPE performs rather poorly for non-uniform record distributions. In the last section we have given a brief introduction to the quantile method which guarantees that MOLHPE with the quantile method exhibits for non - uniform distributions practically the same ideal performance as MOLHPE for uniform distributions.

References

[Bur 83] Burkhard, W.A.:'Interpolation - based index maintenance', BIT 23, 274 - 294 (1983)

[Hin 85] Hinrichs, K.:'The grid file system: implementation and case studies of applications',Ph.D. Dissertation, Swiss Federal Institute of Technology, Zürich (1985)

[Kri 84] Kriegel, H.P.: 'Performance comparison of index structures for multi-key retrieval', Proc. 1984 ACM/SIGMOD Int. Conf. on Management of Data, 186 - 196

[KS 86] Kriegel, H.P., Seeger, B.: 'Multidimensional dynamic quantile hashing is very efficient for non - uniform record distributions', submitted for publication

[KW 85] Krishnamurthy, R., Whang, K.-Y.: 'Multilevel grid files', Draft Report, IBM Research Lab., Yorktown Heights (1985)

[Lar 80] Larson, P.-A.: 'Linear hashing with partial expansions', Proc. 6th Int. Conf. on VLDB, 224 - 232 (1980)

[Lit 80] Litwin, W.: 'Linear hashing: a new tool for file and table addressing', Proc. 6th Int. Conf. on VLDB, 212 - 223 (1980)

[NHS 84] Nievergelt, J., Hinterberger, H., Sevcik, K.C.: 'The grid file: an adaptable, symmetric multikey file structure', ACM TODS, 9, 1, 38 - 71 (1984)

[Oto 84] Otoo, E.J.: 'A mapping function for the directory of a multidimensional extendible hashing', Proc. 10th Int. Conf. on VLDB, 491 - 506 (1984)

[Oto 85] Otoo, E.J.: 'Symmetric dynamic index maintenance scheme', Proc. of Int. Conf. on Foundations of Data Org., 283 - 296 (1985)

[Oto 86] Otoo, E.J.: 'Balanced multidimensional extendible hash tree', Proc. 5th ACM SIGACT/SIGMOD Symp. on PoDS, (1986)

[Ouk 85] Ouksel, M.: 'The interpolation based grid file', Proc. 4th ACM SIGACT/SIGMOD Symp. on PoDS, 20 - 27 (1985)

[Rob 81] Robinson, J.T.: 'The K-D-B-tree: a search structure for large multidimensional dynamic indexes', Proc. 1981 ACM/SIGMOD Int. Conf. on Management of Data, 10 - 18 (1981)

[RL 82] Romamohanarao, W., Lloyd, J.: 'Dynamic hashing schemes', Comp. J., 25, 4, 478 - 485 (1982)