Techniques for Design and Implementation of Efficient Spatial Access Methods *

Bernhard Seeger Hans-Peter Kriegel

University of Bremen, D-2800 Bremen, West Germany

Abstract

In order to handle spatial data efficiently, as required in computer aided design and geo-data applications, a database management system (DBMS) needs an access method that will help it retrieve data items quickly according to their spatial location. In this paper we present a classification of existing spatial access methods and show that they use one of the following three techniques: clipping, overlapping regions, and transformation. From a practical point of view we provide a tool box supporting simple design of a spatial access method for a given point access method using one of the above techniques. We analyze the technique of transformation in more detail and show that our new concept of asymmetric partitioning is more retrieval efficient than the traditional symmetric approach. Furthermore we suggest a hybrid method combining the techniques of overlapping regions and transformation and provide an analysis and comparison of our new method. For data for which an analysis of R- and R^+-trees was available, these comparisons demonstrate a superiority of our scheme.

1 Introduction

Access methods for secondary storage which allow efficient manipulation of large amounts of records are an essential part of a database management system (DBMS). In traditional applications, objects are represented by records, which are d-dimensional points, d $\geq$ 1, and thus point access methods (PAMs) are required. We distinguish access methods for primary keys (one-dimensional points) and access methods for secondary keys (multidimensional points).

However, it turned out that PAM are not sufficient for applications like computer aided design ([SRG 83], [MT 83]), automatic generation of maps [MOD 87], or image processing [RL 85]. In particular, new access methods are necessary for the organization of multidimensional spatial objects, like rectangles, polygons, circles, etc. . We call these methods spatial access methods (SAMs). Additionally, queries asking for spatial objects seem to be more complex than queries asking for points. For instance a typical spatial query is the point-query: Given a point, find all spatial objects that contain the point.

In this paper we will deal with SAMs based on multidimensional dynamic hashing schemes (MDHs). We will show that MDHs without directory, whose most efficient representative is PLOP-Hashing [KS 88], can easily be extended to very efficient SAMs. Moreover, our new method combines different techniques of various previous SAMs. Particularly, our scheme generalizes on one hand the concept of transformation of spatial objects [NH 85], on the other hand the basic concept of R-trees [Gut 84].

In the following we assume that d-dimensional spatial objects are in the d-dimensional unit cube $E^d = [0,1)^d, d \geq 1$. Obviously, this can easily be fulfilled by simple transformation. The problem of storing d-dimensional spatial objects can be reduced to handle d-dim. rectangles by finding the minimum bounding rectangle (MBR) of a spatial object. Moreover we will assume that the sides of the MBRs are parallel to the axis of the data space E^d.

The remainder of this paper is organized as follows. In section 2 we briefly review one of the most efficient MDH schemes without directory, called multidimensional dynamic piecewise linear order preserving hashing, in short PLOP-Hashing. For a more complete discussion we refer to [KS 88]. Then in section 3 we will present a classification of existing SAMs based on three techniques: clipping, overlapping regions, and transformation. We discuss the properties of schemes using these techniques. In section 4 we apply the technique of overlapping regions to PLOP-

[1]This work was supported by grant no. Kr670/4-1 from the Deutsche Forschungsgemeinschaft (German Research Society)

Hashing. In section 5 we apply the transformation and overlapping region technique to PLOP-Hashing which results in a hybrid SAM. The most important contribution of this paper is the design of an asymmetric partitioning and an analysis thereof. Section 6 concludes the paper and gives some aspects of future work.

2 Piecewise Linear Order Preserving Hashing

The basic idea of multidimensional PAMs is to divide the data space into disjoint regions. The objects contained in one region are stored in one bucket, or in a short chain of buckets. In order to support a dynamic adaptation, the number of disjoint regions depends on the number of records. MDH schemes commonly partition the data space using a dynamic grid.

In the past few years a large spectrum of MDH schemes was proposed. MDH schemes fall in one of two categories: those that do not use a directory and those that use a directory. There is a large variety of MDH schemes with directory, like the grid file [NHS 84], multidimensional extendible hashing [Oto 84] or different types of hash trees ([Oto 86], [Ouk 85], [WK 85]). In all these schemes the directory resides fully or partially on secondary storage.

In this section we will review PLOP-Hashing, a MDH scheme without directory. In [KS 88] we presented PLOP-Hashing in detail and reported on an experimental performance comparison with the grid file [NHS 84], where PLOP-Hashing turned out to be the superior scheme.

MDH schemes without a directory are based on (one-dimensional) linear hashing [Lit 80]. Using a hashing function H, we compute the address of a short chain of buckets, where the set of addresses {0,..,m-1} is time varying. Giving up the directory we have to allow overflow records, i.e. records which cannot be placed in the first bucket of the corresponding chain, called primary bucket. The overflow records are stored in a so-called secondary bucket which is chained with the primary bucket. The primary bucket resides in the primary file, the secondary bucket in the secondary file. This very simple type of treating overflow records is called bucket chaining. One chain of buckets is also called a page.

Two basically different order preserving address functions have been suggested for MDH schemes without directory. One of them is the interpolation function [Bur 83] which generates a one-dimensional key from a d-attribute composite key using z-ordering [OM 84] and then computes the address using a one dimensional order preserving hashing function. The other address function is the one used

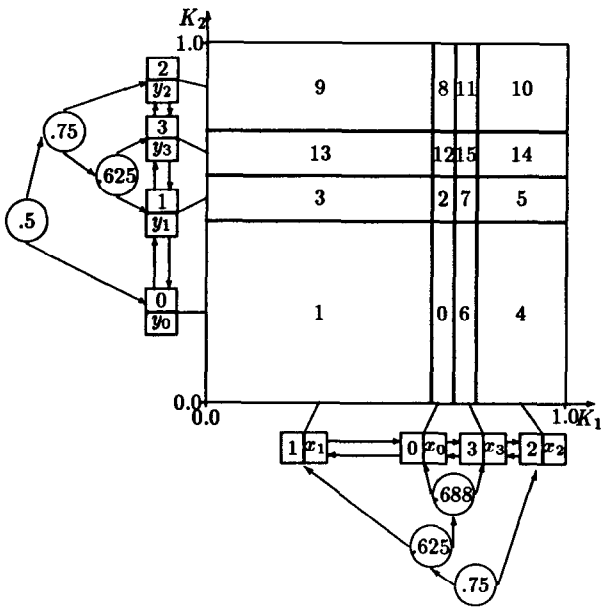

Figure 1: Partition of the data space $[0,1)^2$ generated by PLOP-Hashing

in MOLHPE [KS 86] and quantile-hashing [KS 87] which was originally suggested to compute directory addresses in multidimensional extendible hashing [Oto 84]. In our approach we will use this address function to compute page addresses.

As already indicated, the data space is partitioned by an orthogonal grid, see figure 1. The partitioning points of the grid on each axis are defined by d binary trees, $d \geq 1$, comparable to the scales of the grid file. Each inner node of such a binary tree stores a partitioning point representing a (d-1)-dimensional hyperplane that cuts the space into two rectangular shaped regions. Each leaf is associated with a d-dimensional slice S(i,j) of the data space which is bounded by two neighboring partioning hyperplanes, $0 \leq i < m_j, 1 \leq j \leq d$, where m_j is the number of slices corresponding to the j^{th} axis. Such a slice S(i, j) is addressed by the index i stored in the corresponding leaf, $0 \leq i < m_j$, $1 \leq j \leq d$. The whole data space is the union of d-dimensional rectangles which are not cut by any partitioning hyperplane and are therefore called cells. All the d-dimensional points lying in one cell are stored in one page. The address of that page is computed using the index i_j of all slices $S(i_j, j)$, whose intersection results in the corresponding cell, $0 \leq i_j < m_j, 1 \leq j \leq d$. In figure 1, the addresses of the pages are depicted in the corresponding cells. For example key K = (0.2,0.8) belongs to slices S(1,1) and S(2,2), and therefore K is stored in the page

with address 9. Additionally to an index i, each leaf contains the number x_i^j of points which are in the slice S(i, j), $0 \leq i < m_j, 1 \leq j \leq d$. This information is used to expand the file in a piecewise linear fashion.

Before explaining the principle of piecewise linear expansions, let us introduce some further notations. The file size m is given by the number of pages in the file. The level $L = \lfloor \log_2 m \rfloor$ indicates how often the file size has doubled, assuming a file size of one page at the beginning. Moreover the level l_j of axis j, which is given by $l_j = \lfloor \log_2 m_j \rfloor$, $1 \leq j \leq d$, specifies how often the number of slices of axis j has doubled. For example in figure 1 the file has 16 pages and therefore the level L is 4, $l_1 = 2$ and $l_2 = 2$. During one doubling of the file size the number of pages increases from 2^L to $2^{L+1} - 1$. During this process, one axis s, $1 \leq s \leq d$ called split axis is selected in which the expansions are carried out. Let us assume that the split axis s is chosen in a cyclic order, i.e. s = L MOD d + 1. Then our scheme partitions the data space symmetrically, i.e. $|l_i - lj| \leq 1$, $1 \leq i, j \leq d$.

The pages of the file are arranged in groups $g_0, .., g_{m_s-1}$, where the union of the cells of one group g_j is the slice S(j,s), $0 \leq j < m_s$. A rule, called control function, triggers the expansion of the file by another group of pages, respectively slice. First one group of pages will be selected by the control function (e.g. if the load factor of this group is more than 100%). We have illustrated this situation on the left side of figure 2, where we have a file of 4 pages. Thus the level L = 2 and s denotes the first axis. Now we consider two groups $g_0 = \{0,2\}$ and $g_1 = \{1,3\}$. Assuming $x_0 > 2b$ (b = capacity of a bucket), i.e. the group g_0 contains more than 2b records, the control function calls for an expansion of the file, particularly of the group g_0. Thus the slice S(0,1) will be cut into two by inserting a new partitioning point into the binary tree of the first axis and expanding the file by the group $g_2 = \{4,5\}$, see the right side of figure 2.

The expansion of one group does not proceed in one macro step, but step by step using an expansion pointer $(ep_1, .., ep_d)$, which indicates the page to be expanded next. Thus the split group will be expanded linearly as for linear hashing [Lit 80], and the whole file will be expanded in a piecewise linear fashion.

In addition to the expansion of the file we have considered contraction and reorganization. As for expansion we have a control function, which triggers merging of two groups belonging to neighboring slices into one group. We have chosen the following control function for contraction: Merge the pair of neighboring groups with the minimum number of records, if the load factor is below 45%.

Until now, we have described PLOP-Hashing as a symmetric MDH scheme without directory. However, PLOP-Hashing allows a dynamic asymmetric partitioning of the data space. In this sense "dynamic" means that after doubling of the file size the expansion axis can be chosen in an arbitrary way yielding an asymmetric partition. Moreover considering an expansion of the file, PLOP-Hashing allows distributing the records of c slices over c+1 slices, $c > 0$, and thus supports partial expansions ([Lar 80], [KS 86]).

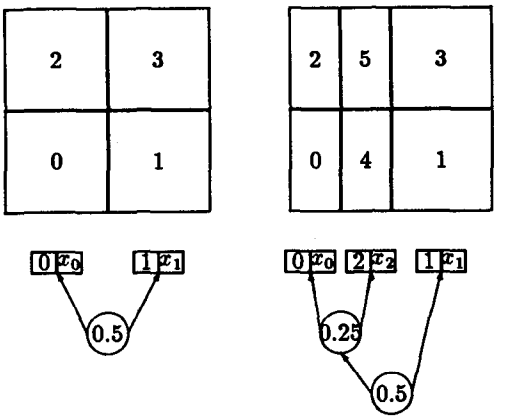

Figure 2: Expansion of a file organized by PLOP-Hashing

3 An overview of spatial access methods

In this section we will provide an overview of spatial access methods (SAMs) which are based on the approximation of a complex spatial object by the minimal bounding rectangle (MBR) with the sides of the rectangle parallel to the axes of the data space. In figure 3, polygons are illustrated together with their MBRs in the 2-dimensional data space $[0,1)^2$. The most important property of this simple approximation is that a complex object is represented by a limited number of bytes. Although a lot of information is lost, MBRs of spatial objects preserve the most essential geometric properties of the object, i.e. the location of the object and the extension of the object in each axis.

There are other proposals using more complex approximations of spatial objects. The most interesting of them, which is particularly designed for the organization of data in secondary memory, is used in the PROBE-project [Ore 86]. Similar to quad- and octtrees [Sam 85], a grid is assumed covering the spatial object where all cells of the grid intersecting the object are stored in a file. Combining cells to rectangles and applying z-order [OM 84] to identify these rectangles, results in a more compressed representation of objects than in case of octtrees. To support clustering of

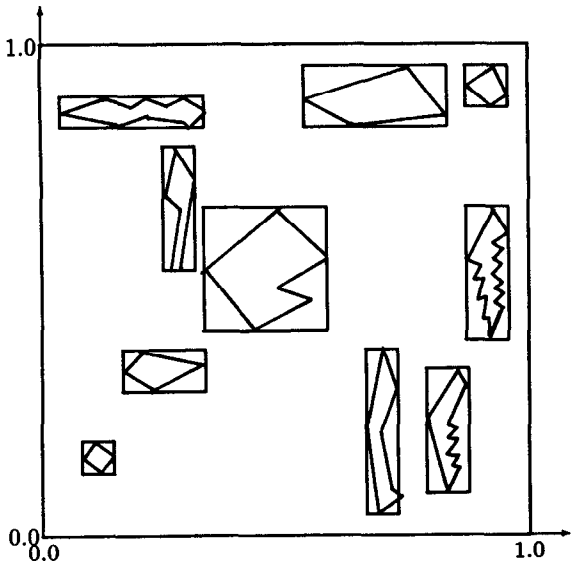

Figure 3: Some polygons and their corresponding MBRs

spatial objects, all corresponding rectangles of the object are stored in a separate file for this particular object. A minimal bounding rectangle characterizes an object using a short record. Thus it is possible to organize the approximation of different objects in a single file. For every minimal bounding rectangle of an object, a pointer refers to a file where a more exact approximation or the exact description of the object is stored.

SAMs organizing minimal bounding rectangles of objects can be classified into three groups. Each of these groups is characterized by a special technique that allows an extension of a multidimensional point access method (PAM) to a multidimensional SAM. Thus the performance of such SAMs depends on the underlying PAM and depends on the applied technique. In the remainder of this section we will essentially limit our discussion to the properties of these techniques.

In the following, we consider several so-called spatial queries. The first two queries should be supported by each SAM :

1. point query :
   Given a point $P \in E^d$, find all d-dim. rectangles R in the file with $P \in R$
2. rectangle intersection :
   Given a d-dim. rectangle $S \subseteq E^d$, find all d-dim. rectangles R in the file with $S \cap R \neq \emptyset$
3. rectangle enclosure :
   Given a d-dim. rectangle $S \subseteq E^d$, find all d-dim. rectangles R in the file with $R \supseteq S$
4. rectangle containment :
   Given a d-dim. rectangle $S \subseteq E^d$, find all d-dim. rectangles R in the file with $R \subseteq S$
5. volume queries :
   Given $v \in (0,1)$, find all d-dim. rectangles R in the file with volume equal to v.

As demonstrated in [HKSS 88], queries (3) - (5) are very important to recognize similar CAD-objects in CAD data. In order to maintain low insertion costs, exact match queries should also be supported efficiently. In particular, a SAM should be dynamic, i.e. insertions and deletions should not reduce retrieval performance. In analogy to PAMs it is important that retrieval performance should essentially be independent of the distribution of the spatial objects. These last two aspects usually depend on the underlying PAM and are therefore not discussed in detail in this section.

Additionally to the distribution of objects the density O(P) of a point $P \in E^d$ can influence retrieval performance, where the density O(P) is the number of rectangles in the file containing $P \in E^d$. The global density of a file is given by

$$O = \max_{P \in E^d} O(P) \qquad (1)$$

The value of O heavily depends on the particular application. Let us consider two applications in the area of cartography, where we organize polygons using the minimal bounding rectangles (MBR) of the polygons:

1. storing contour lines, which are used in topographical maps, leads to a high global density
2. storing limits of lots, which are used in conventional maps, commonly leads to a low density

3.1 Clipping

Clipping can easily be explained by describing the insertion of a new rectangle. Assuming a partition of the data space into disjoint regions, an insertion of a rectangle will be performed like an insertion of a point. Problems will only occur, if a rectangle R intersects with more than one disjoint region. Clipping of a rectangle means that R is partitioned into a minimal set of rectangles $\{R^1, .., R^q\}$, where

$$R = \bigcup_{i=1}^{q} R^i, \; q > 1$$

Every rectangle $R^i, 1 \leq i \leq q$, intersects with exactly one disjoint region. Now we can insert these q rectangles $R^1, .., R^q$ into the file.

In figure 4 we have depicted the partition of the data space after insertion of ten 2-dimensional rectangles $R_1, .., R_{10}$

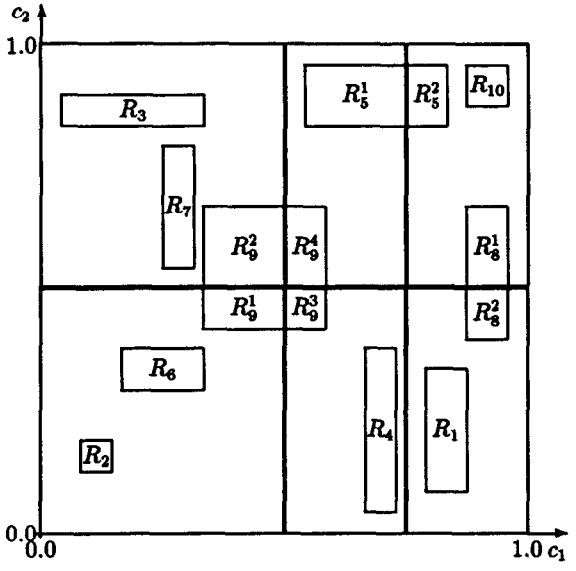

Figure 4: Insertion of ten 2-dim. rectangles $R_1, .., R_{10}$ into a file using clipping based on PLOP-Hashing

into the file using clipping based on PLOP-Hashing. The file consists of six pages, three slices vertical to axis 1 and two slices vertical to axis 2. Rectangles R_5 and R_8 are partitioned into two, rectangle R_9 is partitioned into four rectangles.

The most important advantage of schemes using clipping is that they are really extensions of the underlying PAM. If such a scheme organizes only point data, it will inherit all the properties of the underlying PAM. Additionally, d-dim. points and d-dim. rectangles can be organized together in one file.

However, a drawback is obviously the duplication of rectangles in the file, for instance rectangle R_9 must be stored four times in our example, see figure 4. Thus storage utilization will be indirectly reduced. Deletions and insertions require more disk accesses than in case of point data. In particular, a rectangle must be inserted in all buckets where the corresponding region intersects the rectangle. Moreover, assuming a bucket capacity of b records, $b > 1$, and an underlying point access method allowing no overflow records, all such SAMs based on clipping require that the following condition 2 is fulfilled

$$O = \max_{P \in E^d} O(P) \leq b \qquad (2)$$

This phenomenon can be easily illustrated by an example. Let us assume b=2 and three 2-dim. rectangles $R_1 \cap R_2 \cap R_3 \neq \emptyset$. Because overflow records are avoided, these rectangles must be stored on several buckets. In particular, a hyperplane must exist that separates these rectangles into two subsets. This hyperplane requires clipping at most one of these rectangles. Therefore at least on the left or on the right of the hyperplane, there will be again three rectangles with a nonempty common intersection.

PLOP-Hashing allows overflow records and therefore the functionality of the derived SAM is guaranteed. However, retrieval performance can suffers from the redundance introduced by clipping.

Assuming condition 2 is fulfilled, let us now consider spatial queries. To answer a point query (1) and a rectangle enclosure query (3) only one data page must be accessed. Obviously, this behavior makes such schemes very attractive. Assuming the same search rectangle for query 2 and query 4 the same data buckets must be accessed (and thus the same number of disk accesses is required) although the number of answers of query 4 is usually much lower than of query 2. In case of query 4, a lot of records, so-called false drops, must be accessed which do not satisfy the query. To reduce disk accesses for these queries, rectangles that are completely contained in an arbitrary query rectangle must be separated from all other rectangles with a nonempty intersection outside of the query rectangle. If this requirement is fulfilled, search regions will be disjoint and thus the number of false drops can be reduced. Let us now consider query 5. This type of query cannot be supported in anyway by a SAM based on clipping, i.e such a query can only be answered, if all data pages in the file are accessed.

It is important to realize that clipping can be applied to every multidimensional PAM. In particular, the R^+-tree [SRF 87], Box - Excell [TS 82] and the multi layer grid file [SW 88] are SAMs applying the technique of clipping based on the K-D-B-tree [Rob 81], Excell [TS 82] and the grid file [NHS 84], respectively. All these underlying PAMs avoids overflow records and therefore the functionality of the corresponding SAM is limited to applications where condition 2 is fulfilled.

3.2 Overlapping Region Schemes

Such as clipping, overlapping region schemes (OR - schemes) organize d-dimensional rectangles using a d-dimensional PAM. For the following considerations we define the region of a bucket as the minimal bounding box of the rectangles belonging to the bucket. Contrary to clipping, OR-schemes allow data buckets where the corresponding regions have a common overlap. We will discuss the principle of OR-schemes by a short introduction to the R-tree [Gut 84], one of the most popular SAMs.

The R-tree is a balanced tree generalizing the B^+-tree concept [Com 79] to spatial objects. Storage utilization is

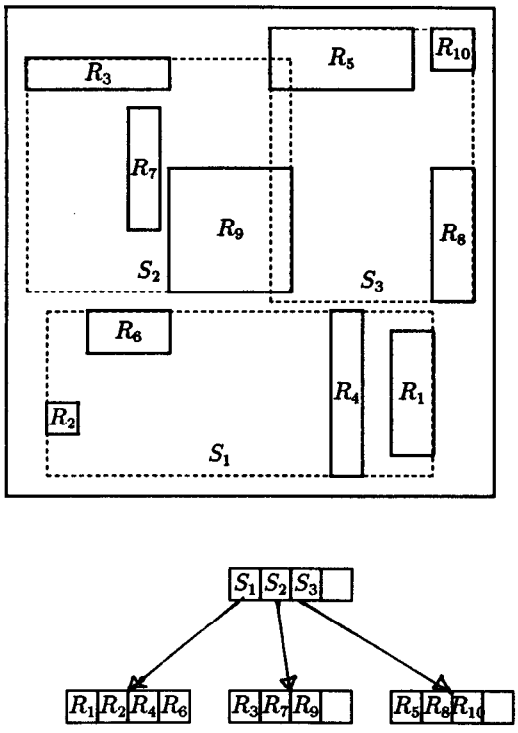

Figure 5: Organization of some rectangles and the corresponding structure of the R-tree

guaranteed to be above 50%. Minimal bounding rectangles of spatial objects are stored in the leaves of the tree, where each of the leaves corresponds to a data bucket. In an inner node of the tree there are tuples (R,p), where p is a pointer referring to a son and R is the minimal bounding rectangle of all rectangles in the corresponding son. Since clipping of rectangles is avoided, a rectangle is stored in exactly one of the data blocks. Thus overlapping regions of different data blocks are allowed for the organization of spatial objects.

In figure 5 we have illustrated the structure of an R-tree and the partition of the corresponding data space. As demonstrated, the regions S_2 and S_3 have a non-empty common intersection.

The advantage of OR-schemes is that storage utilization depends only on the underlying PAM, since every rectangle is uniquely represented in the file. Thus the B^+-tree inherits the guarantee of at least 50% storage utilization to the R-tree. Another nice property is that, in analogy to clipping methods, d-dim. points and d-dim. rectangles can be organized together in one file.

However, retrieval performance depends heavily on the amount of overlap. As demonstrated in [FSR 87], retrieval performance can degenerate, if rectangles with highly varying volumes occur. Particularly to answer an exact match query more than one access to data buckets is usually required. Obviously, this will increase costs for insertions and deletions.

Let us now consider spatial queries as introduced at the beginning of this section. Depending on the amount of overlap, a point query requires more than one access to a data bucket and quite a few accesses to directory buckets. For example, assuming a point $P \in S_2 \cap S_3$ in figure 5, the corresponding point query requires access to two data buckets. Similar to clipping schemes, both, a rectangle containment query and a rectangle intersection query require access to the same data buckets. As mentioned, a separation of these different rectangles can improve performance.

In [Ooi 87] overlapping regions is proposed for the kd-tree [Ben 75]. In section 4 we will to apply the technique of overlapping regions to PLOP-Hashing.

3.3 Transformation

The basic idea of transformation-schemes (T-schemes) is to represent minimal bounding rectangles of multidimensional spatial objects by higher dimensional points. For instance, a 2-dimensional rectangle R with sides parallel to the axis is represented by a 4-dimensional point (center representation)

$$(c_1, c_2, e_1, e_2)$$

where $c = (c_1, c_2) \in (0,1)^2$ is the center of the rectangle and $e = (e_1, e_2) \in (0, 0.5)^2$ is the distance of the center to the sides of the rectangle. As proposed by Nievergelt and Hinrichs [NH 85], these 4-dimensional points can be organized by the grid file [NHS 84], generally speaking by a multidimensional PAM.

Another choice of parameters is the corner representation, where a 2-dim. rectangle can be represented by its lower left corner $(l_1, l_2) \in [0,1)^2$ and its upper right corner $(u_1, u_2) \in [0,1)^2$. However, the choice of the parameters can influence performance and characteristics of the SAM. The basic advantage of the center representation is that location parameters, like the center of a rectangle, are distinct from extension parameters. Moreover, the center of a rectangle seems to be the best location parameter, since the distance to all other points within the rectangle is minimized.

Due to the illustration of examples, in the following we limit our considerations to segments (1-dim. rectangles). According to the different representations, a segment can be described by (c,e), where $c \in (0,1)$ is the center and $e \in (0, 0.5)$ is half of the length of the segment or by (l, u),

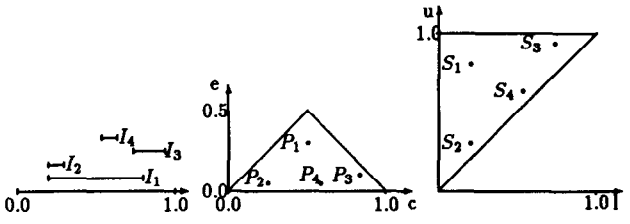

Figure 6: Transformation of segments I_j into 2-dim. points $P_j = (c_j, e_j)$ (center representation) and $S_j = (l_j, u_j)$ (corner representation), j = 1,..,4

where $l \in [0,1)$, and $u_\epsilon(0,1)$ is the left and right limit of the segment, respectively. Let us mention that most PAMs - in particular PAMs based on hashing, such as the grid file [NHS 84], hashtrees [Oto 86] or quantile hashing [KS 87] - assume a rectangular shaped data space. Retrieval performance and storage utilization will decrease, if the data space contains regions where no data occurs, so-called dead regions . Let us consider the corner representation, then the 2-dim. data space must be the unit square $[0,1)^2$. However data occurs only above the diagonal, because $l < u$. Thus in case of segments in half of the data space no data occurs and for 3-dimensional rectangles only 1/8 of the data space can contain data. Using the center representation, the 2-dim. data space is given by $(0,1) \times (0,0.5)$. As depicted in figure 6 data can only occur in the triangular shaped subspace

$$T = \{(c,e) \mid e < \min(c, 1.0 - c), c \in (0,1.0)\}$$

However, since in common applications the length of segments is quite short with respect to the unit segment [0,1), the subspace T can be reduced to a trapezoid formed subspace (also called the real data space)

$$T_{emax} = \{(c,e) \mid e < \min(c, 1.0 - c, emax), c \in (0,1.0)\}$$

where $emax \in (0,0.5)$ is the maximum length of a segment presently in the file. For $emax \approx 0$ the real data space corresponds approximately to the rectangular shaped data space $(0,1) \times (0, emax)$.

Assuming segments, spatial queries can be illustrated in 2-dim. data space. For a segment S, we have depicted in figure 7 the disjoint regions where segments R occur with $R \supseteq S$, $R \subseteq S$ and $R \cap S \neq \emptyset$. Obviously, T-schemes have the advantage that the rectangle enclosure query is particularly supported, i.e all answers for the query are in a cone shaped region of the data space $(0,1) \times (0, emax)$. In a similar way, the search region of point queries, rectangle intersection queries and rectangle containment queries can be represented as cone shaped subspace of the data space,

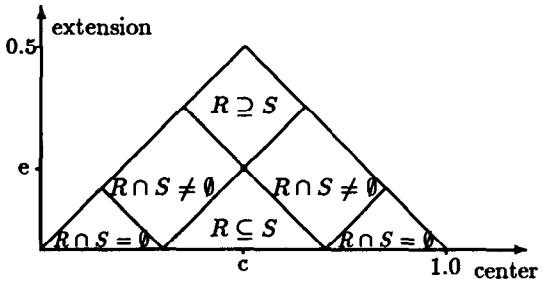

Figure 7: Assuming the center representation and a segment S=(c,e), the regions where are all segments R with $R \supseteq S$,$R \subseteq S$ are disjoint, and the region where are all segments R with $R \cap S \neq \emptyset$ contains the other "search" regions

see figure 7. An additional advantage of T-schemes is that queries are supported asking for the volume of rectangles or for the length of the sides of rectangles.

In analogy to OR-schemes, rectangles are uniquely represented in the file. Applications are not restricted to those fulfilling a particular condition such as for clipping schemes. Additionally, all properties of the underlying PAM are inherited to the T-scheme.

The main drawback is that d-dimensional rectangles lying close together are spread out in the 2d-dimensional data space. This will typically occur if rectangles have strongly varying volumes. However, a large distance of segments in the transformed data space does not imply that these rectangles are stored far away from each other in the file.

The major problem of T-schemes is how we can estimate the value of emax to reduce the data space as much as possible. For a motivation of the problem let us consider the grid file [NHS 84]. During the initialization of the grid file the user must fix the data space for the whole life cycle of the file. At the point of initialization, the user often has no knowledge of the data to arrive later. Thus, typically the value of emax will be highly overestimated or even emax will be set to the possible maximum value of the domain. In the following sections, we will demonstrate how we can avoid these drawbacks of existing T-schemes using PLOP-Hashing and the transformation technique.

4 Overlapping Regions applied to PLOP-Hashing

In this section we will propose a SAM based on PLOP-Hashing and the technique of overlapping regions. As mentioned in section 2, PLOP-Hashing organizes the data

space using a dynamic grid. The d-dim. grid is specified by d binary trees which reside in main memory. In the leaves of the binary trees we store information to compute addresses and additionally we store the number of records in a slice to control expansion. Let us consider 2-dim. rectangles given by

$$(c_1, c_2, e_1, e_2)$$

where $c = (c_1, c_2) \in (0,1)^2$ is the center of the rectangle and $e = (e_1, e_2) \in (0, 0.5)^2$ is the distance of c to the sides of the rectangle. The center of the rectangle uniquely determines the address of the page storing the rectangle. The address is independent of the distance e. We will see that e determines the degree of overlap.

Additionally to PLOP-Hashing used as a PAM, we store in the leaf of the j-th binary tree corresponding to slice S(i,j) the minimum min(i,j) and the maximum max(i,j), $0 \leq i < m_j$, $1 \leq j \leq d$, where

$$\min(i,j) := \min \{l \mid l = c_j - e_j, R = (c,e) \textit{ is a rectangle in the file with } c \in S(i,j)\}$$
$$\max(i,j) := \max \{l \mid l = c_j + e_j, R = (c,e) \textit{ is a rectangle in the file with } c \in S(i,j)\}$$

This additional information is stored in the leaf corresponding to the slice S(i,j). To give an intuitive understanding of this method, we will consider the example illustrated in figure 8. The data space is divided by the grid in six disjoint regions, each corresponding to a page on secondary storage. In the snapshot depicted in figure 8 we have inserted 10 rectangles R_1,..,R_{10} in the file, the same rectangles as in the example of section 3.1 . Although rectangle R_9 intersects 4 grid cells, its address is determined by its center and thus R_9 is stored in the same page as R_3.

Now let us consider insertion of the rectangle $R_{11} = (c_{11}, e_{11})$, where $c_{11} = (0.55, 0.35)$ and $e_{11} = (0.15, 0.05)$. We proceed as follows:

1. Searching the binary trees using the components of c_{11} yields the 2-dim. index $i_1 = 1$ (see figure 8) and $i_2 = 0$, and thus yields the address of the page, where the rectangle has to be inserted.
2. Since $0.4 = c_{11} - e_{11} < min(1,1) = 0.588$, min(1,1) has to be updated (min(1,1) := 0.4).
3. Additionally max(0,2) has to be updated (max(0,2) := 0.4).

Similar to the R-tree, the cost of a point query is usually more than one disk access. Considering performance of exact match queries, insertions and deletions, in our method one page access suffices whereas the number of

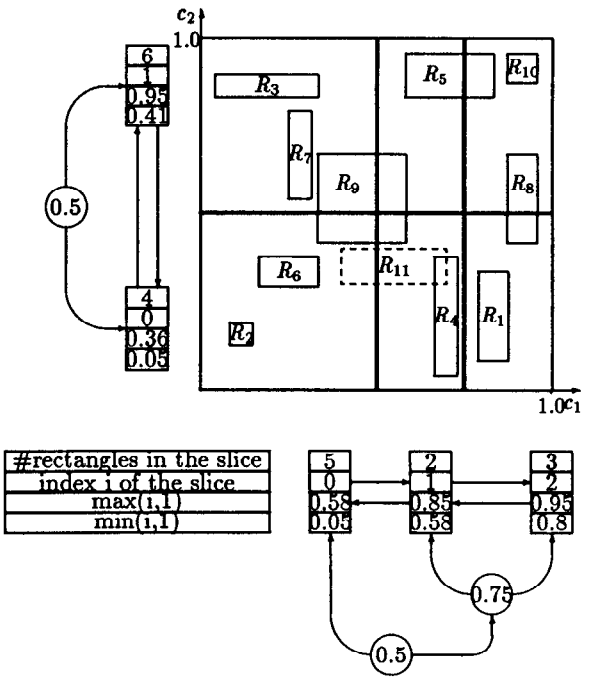

Figure 8: OR-method applied to PLOP-Hashing with the binary trees shown

disk accesses in the R-tree increases with increasing size of the rectangle. As in all schemes applying the OR-method, the performance of point and rectangle intersection queries in our scheme depends on the variation in the size of the rectangles. Insertion of some large rectangles may reduce the performance rapidly. We conclude that a combination of the technique of overlapping regions and a PAM based on an efficient MDH scheme is an interesting competitor to the R-tree.

5 Asymmetric partioning of the data space

In this section we will propose a variant of PLOP-Hashing suitable for organization of 2d-dimensional points, which are generated by transformation of d-dim. rectangles. Nievergelt and Hinrichs [NH 85] have proposed a similar scheme based on the grid file. However our scheme offers three essential improvements: the partition of the 'real' data space, the dynamic organization of each axis and the asymmetric partition of the data space. These three properties are discussed in the remainder of this section. We want to emphasize that property 2 and to a large portion property 1 cannot be achieved in a MDH scheme with directory, like the grid file.

In the following, we consider the organization of segments (one-dimensional rectangles) transformed into 2-dim. points (c,e) using the center representation, where c is the center of the segment, $0 < c < 1$ and e is the distance of c to the margin, $0 < e < 0.5$. As shown in section 3.3, we can reduce our data space to

$$T_{emax} = \{(c,e) \mid 0 \leq c \leq 1.0,\ 0 \leq e \leq emax\}$$

To maintain a dynamic organization of the c-axis, when the value of emax changes, we store for each slice of the c-axis the maximum e-value

$$emax_i = \max \{e \mid I = (c,e) \textit{ is a segment in the file with } (c,e) \in S(i,c)\}$$

where S(i,c) is the slice for index i in the c-axis, $0 \leq i < m_c$ and m_c is the number of slices in the c-axis. Then emax is given by $emax = \max \{emax_i \mid 0 \leq i < m_c\}$.

We emphasize that in case of the grid file the domain of the scales is fixed. The maximum and minimum of the domain must be chosen during the initialization of the file. Moreover the values of records which will be inserted in the file are unknown. To guarantee the functionality of the method, the maximum of the e-axis is highly overestimated. Thus in the grid file empty data space where records do not occur is likely to be created.

The knowledge of emax, the maximum in the e-axis, heavily influences the type of partitioning of the data space. PLOP-Hashing was proposed to partition the data space symmetrically (see section 2). Nevertheless, allowing asymmetric partitions will improve retrieval performance. Let us consider an example, where $emax \approx 0$. Since segments are nearly reduced to one-dimensional points, a scheme which partitions only the c-axis (such as proposed in section 4), seems be more attractive than a scheme with a symmetric partition of the data space.

5.1 Choice of the partition in case of uniform distribution

In the following, our goal is to minimize the number of disc accesses in a point query using PLOP-Hashing as underlying PAM. The above considerations indicate that the minimum will generally not be achieved for a symmetric partitioning of the center-axis and the extension-axis, i.e. the levels of both axes differ by at most one (to put it differently, the number of partitioning points in one axis is at most twice the number of partitioning points in the other axis). In this section we will optimize the degree of asymmetry of both axis, i.e. we will optimize levels l_c and l_e of the center-axis and extension-axis, respectively, for a given global level L. We will assume that the 2-dimensional records (c,e) $\in T_{emax}$ are uniformly distributed in the data space T_{emax}. Obviously, this is not a realistic assumption. However, under this assumption we will be able to derive the difference in performance for schemes with symmetric and asymmetric partition of the data space with analytic tools. Under the assumption of uniform distribution, PLOP-Hashing partitions the data space in equidistant cells.

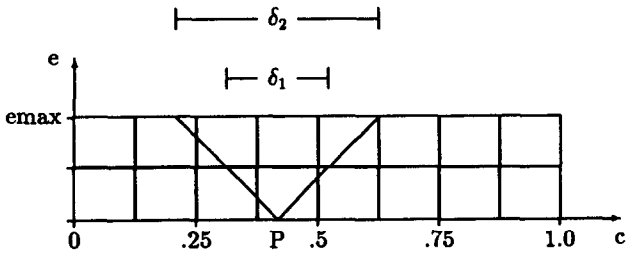

Figure 9: A file with 16 pages and asymmetric partition of the data space T_{emax}, where $L = 4$, $l_c = 3$, $l_e = 1$, $P_c = \{0, 1/8, .., 7/8, 1\}$, $P_e = \{0, emax/2, emax\}$

Let us now consider a file organized by PLOP-Hashing, which consists of n records and 2^L pages, $L \geq 0$, where L is the level of the file. Thus the set P_c of partitioning points of axis c is given by $P_c := \{\ i/2^{l_c} \mid 0 \leq i \leq 2^{l_c}\}$ and the set P_e of partitioning points of axis e is given by $P_e := \{emax * i/2^{l_e} \mid 0 \leq i \leq 2^{l_e}\}$. The variables l_c and l_e denote the level of the axis c and axis e, respectively. Then the level of the file is given by $L = l_c + l_e$. In figure 9 we have illustrated the different terms.

Our goal is to estimate the average number of disk accesses for answering a point query or a rectangle intersection query. In this report, we will only consider point queries. A generalization to more complex queries is obvious.

Let $P \in I_{emax} := [emax, 1.0 - emax)$. We will ask for all segments containing this point P. Since $P \in I_{emax}$, the region where answers can occur is a right-angled triangle. Assuming a uniform distribution the expected number of answers for a point query is $emax * n$. Now we calculate the number of cells which intersect with the search region. Thus we need the lengths δ_i, $1 \leq i \leq 2^{l_e}$, of the line segments, which are obtained by intersection of the search region and the hyperplanes belonging to the partitioning points of the e-axis, see figure 9. Then we obtain $\delta_i = 2i * emax/2^{l_e}$, $i = 0, .., 2^{l_e}$. Thus the expected value E_i how often hyperplanes of axis c intersect the line segments is $E_i = \delta_i * 2^{l_c} = 2i * emax * 2^{l_c - l_e}$, $i = 0, .., 2^{l_e}$. The expected number A of grid cells, which intersect the search region is given by $A = \sum_{i=1}^{2^{l_e}} (E_i + 1) = emax * (2^{l_c} + 2^L) + 2^{l_e}$. For

$l := l_c$, we obtain the formula depending on the variable l:

$$A(l) := emax * (2^l + 2^L) + 2^{L-l} \quad 0 \leq l \leq L \qquad (3)$$

The minimum value l_{min} of the function A(l) can be computed using the derivation A'(l). Then we obtain by A'(l_{min}) = 0

$$l_{min} = \begin{cases} L & if emax * 2^L < 1 \\ (L - \log_2 emax)/2 & \text{otherwise} \end{cases} \qquad (4)$$

The minimum number $A_{min} := A'(l_{min})$ of grid cells which intersect the search region is given by

$$A_{min} = \begin{cases} 2emax * 2^L + 1 & if emax * 2^L < 1 \\ emax * 2^L + 2(emax * 2^L)^{1/2} & \text{otherwise} \end{cases} \qquad (5)$$

Thus the number of disk accesses to answer a point query is given by $A(l)*cl$, where cl denotes the average number of buckets per chain (page). To represent our result (5) as a function of n, we derive the usual performance measure pm for a complex query which is given by the quotient of the average number of accessed candidates (including the false drops) and the average number of answers. Considering the average storage utilization $su = n/(2^L * b)$, we obtain for $2^L * emax \geq 1$

$$pm = \frac{(A_{min} * cl) * (b * su)}{emax * n} = cl + O(1/\sqrt{n}) \qquad (6)$$

after insertion of formula (5) and further manipulations.

This result is independent of the degree of overlap. The first term of the sum corresponds to the average chain length which will be almost 1 in case of uniform distribution. The second term express the additional overhead introduced by the margin of a search region which is also induced by the margin of a range query in a MDH scheme storing point objects.

In order to compare the difference of the performance for schemes which partition the data space in a symmetric fashion and schemes which partition the data space in a optimal way, we will present some analytic results.

In the following we assume a file of 2^L, $L \geq 0$, pages, which are completely filled. Thus the number n of records in the file is $n = 2^L * b$, where b is the capacity of a bucket. To compare the performance, we define the parameter $VAR = (A_{sym} - A_{min}) / A_{sym}$, where $A_{sym} = A(L/2)$ is the number of cells intersecting the search region in case of a symmetric partition.

In our first diagram of the appendix (figure 10), we have depicted A_{sym} and A_{min} depending on the level L of the file, where $b = 50$ and emax = 1/256. The difference in performance is essential. After insertion of 51200 segments, 8 grid cells intersect the search region, whereas for a symmetric partition 36 grid cells intersect the search region. In figure 11 of the appendix, we have depicted the parameter VAR depending on the number of records.

In [FSR 87] the performance of R-trees and R^+-trees is analyzed for a special uniform distribution. Due to space limitations this distribution is not explained here. We have evaluated the performance in case of this distribution for our scheme and depict it in figure 12 and 13 of the appendix. In figure 12 the number of segments n is fixed (100,000) and the length of the segments and thus the density O is varying. In figure 13 the density O is fixed (O=40) and the number of segments is varying. Both figures show the number of disk accesses to answer a point query. These figures, which were originally presented in [FSR 87] without the values of our scheme, demonstrate the superior behavior of our scheme compared to R-trees and R^+-trees.

These results demonstrate that for an efficient SAM based on transformation, we have to allow for asymmetric partitions. As mentioned before, a uniform distribution of segments is unlikely to occur in practice. Thus the results of this section are more of a theoretical nature.

5.2 Choice of the partion in case of non-uniform distributions

In this section we do not assume an uniform distribution as in section 3.1. Thus we cannot use formula (3) for deciding, how we should partition the data space T_{emax}.

Let us now assume that the file consists of 2^L pages and let us assume we have to decide, which axis should be the next split axis. The grid partition GP is given by

$$GP := (P_c, P_e) := \{ (c,e) \mid (c \in P_c \wedge 0 \leq e \leq 0.5) \vee (e \in P_e \wedge 0 \leq c \leq 1) \}$$

where P_c and P_e are the set of partitioning points of axis c and e, respectively. Now we proceed as follows:

1. Compute grid partitions GP_e and GP_c where $GP_e = (P_c, \tilde{P}_e) \mid \tilde{P}_e \mid = 2^{l_c+1} + 1$ and $GP_c = (\tilde{P}_c, P_e) \mid \tilde{P}_c \mid = 2^{l_c+1} + 1$
2. Determine some points $P_1, .., P_k \in [0,1)$, $k > 1$. Compute the number of grid cells A_c^i, A_e^i of the grid partition GP_c, GP_e, respectively, which intersect the search region of P_i, $1 \leq i \leq k$.
3. If $\sum_{i=1}^k A_c^i > \sum_{i=1}^k A_e^i$, then e is the next split axis,otherwise c is the next split axis.

In step 1 the "virtual" grid partitions GP_e and GP_c are constructed from the "real" grid partition GP and additional information in the leaves of the binary trees. The

new sets $\tilde{P}_c$ and $\tilde{P}_e$ depend on the sets P_e and P_c. The sets are generated by an interpolation technique. In step 2 the parameter k is not fixed, but should depend on the number of records in the file. We want to emphasize that this algorithm does not need any disk access. All the information which is used by the algorithm is stored in main memory.

6 Conclusion

The contribution of this paper can be summarized as follows:

- A classification of existing spatial access methods is derived. Every spatial access method is based on a multidimensional point access method applying one of the following three techniques: transformation, clipping and overlapping regions.
- As an example, we applied these different techniques to one of the most efficient point access methods, PLOP - Hashing. Additionally we discuss in detail the technique of transformation and introduce the concept of asymmetric partitioning which is more efficient than the traditional symmetric partitioning.
- We proposed a hybrid method based on PLOP-Hashing combining the techniques of overlapping regions and transformation. This hybrid method improves performance by tuning to the characteristics of the particular application.
- We provide an analysis of the hybrid method in comparison to a scheme which partitions the data space symmetrically. Moreover we present a brief comparison to R-trees and R^+-trees, which demonstrate the superiority of our scheme.

In our future work we will verify our results by experiments in various applications with an implementation of our scheme. Our goal is an experimental comparison of different spatial access methods.

Acknowledgement

We would like to thank the anonymous referees for their valuable suggestions.

References

[Ben 75] Bentley, J.L.:'Multidimensional Search Trees Used for Associative Searching', Communications of ACM, Vol. 18, No. 9,1975,pp.509-517

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

[Com 79] Comer, D.:"The Ubiquitous B-tree", Computing Surveys, Vol. 11, No. 2,1979,pp.121-137

[HKSS 88] Heep, S., Kriegel, H.P., Schneider, R., Seeger, B.: 'Konzepte zur Suche geometrischer Bauteile', Proc. GI Fachgespraech Non-Standard Datenbanken fuer Anwendungen der graphischen Datenverarbeitung, in German

[FNPS 79] Fagin, R., Nievergelt, N., Pippenger, N., Strong, R.:'Extendible Hashing - A Fast Access Method for Dynamic Files', ACM TODS 4(3), 1979

[FSR 87] Faloutsos, C., Sellis, T., Roussopoulos, N.: 'Analysis of object oriented spatial access methods' Proc. ACM SIGMOD Int. Conf on Management of Data, 1987

[Gut 84] Guttman, A.:'R-trees: a dynamic index structure for spatial searching', Proc. ACM SIGMOD Int. Conf on Management of Data, 47-57, 1984

[KS 86] Kriegel, H.P., Seeger, B.:'Multidimensional order preserving linear hashing with partial expansions', Proc. Int. Conf. on Database Theory, 1986, Lecture Notes in Computer Science 243, 203-220

[KS 87] Kriegel, H.P., Seeger, B.:'Multidimensional quantile hashing is very efficient for non-uniform distributions', Proc. Int. Conf. on Data Engineering, 1987, 10-17, extended version will appear in Information Science

[KS 88] Kriegel, H.P., Seeger, B.:'PLOP-Hashing: a grid file without directory', Proc. Int. Conf. on Data Engineering, 1988, 369-376

[Lar 80] Larson, P.Å.: 'Linear hashing with partial expansions', Proc. 6^{th} Int. Conf. on VLDB, 224-232, 1980

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

[MOD 87] Manola, F., Orenstein, J., Dayal, U.:'Geographic information processing in the Probe database system', Proc. 8^{th} Int. Symp. on Automation in Cartography, Baltimore, 1987

[MT 83] Mantyla, M., Tamminen, M.:'Localized set operations for solid modeling', Computer Graphics, 17, 3, 279-288, 1983

[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

[NH 85] Nievergelt, J., Hinrichs, K.:'Storage and access structures for geometric data bases', Proc. Int. Conf. on Foundations of Data Organization, 335-345, 1985

[Ooi 87] Ooi, B.C.:'A data structure for geographic database', Proc. on 2^{nd} GI Conf. on Database Systems for Office Automation, Engineering, and Scientific Application, 1987

[Ore 86] Orenstein, J.A.:'Spatial Query Processing in an Object-Oriented Database System', Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986

[OM 84] Orenstein, J.A., Merett, T.H.:'A class of data structures for associative searching', Proc 3^{th} ACM SIGACT/SIGMOD Symp. on PODS, 1984

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

[Oto 86] Otoo, E.,J.:'Balanced multidimensional extendible hash tree', Proc 5^{th} ACM SIGACT/SIGMOD Symp. on PODS, 1986

[Ouk 85] Ouksel, M.:'The interpolation based grid file', Proc 4^{th} ACM SIGACT/SIGMOD Symp. on PODS, 1985

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

[RL 85] Roussopoulos, N., Leifker, D.:'Direct spatial search on pictorial databases using packed R-trees', Proc. ACM SIGMOD Int. Conf on Management of Data, 17-31, 1985

[Sam 85] Samet, H.:'The Quadtree and Related Data Structures', Computing Surveys, Vol 16, No. 2,1984

[SRF 87] Sellis, T., Roussopoulos, N., Faloutsos, C.: 'The R^+-tree: a dynamic index for multi-dimensional objects', Proc. 13^{th} Int. Conf. on VLDB, 1987

[SW 88] Six, H.-W., Widmayer, P.: 'Spatial Searching in Geometric Databases', Proc. Int. Conf. on Data Engineering, 1988

[SRG 83] Stonebraker, M., Rubenstein, B., Guttman, A.: 'Application of abstract data types and abstract indices to CAD data bases', Proc. ACM SIGMOD Conf. on Engineering Design Applications, 1983

[TS 82] Tamminen, M., Sulonen, R.: 'The Excell method for efficient geometric access to data', Proc. 19^{th} ACM Design Automation , Conf.,345-351,1982

[WK 85] Whang, K.-Y., Krishnamurthy, R.:'Multilevel grid files', draft report, IBM Research Lab., Yorktown Heights, 1985

Appendix

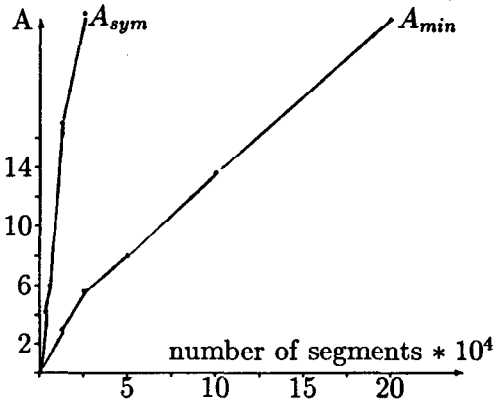

Figure 10: Disk accesses for symmetric and asymmetric partioning depending on the number of segments (b = 50, emax = 1/256)

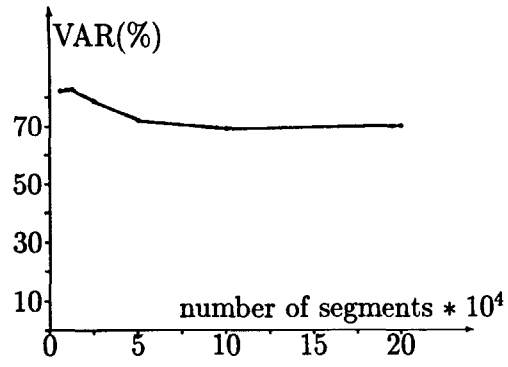

Figure 11: Performance gain VAR depending on the number of segments (b = 50, emax = 1/256)

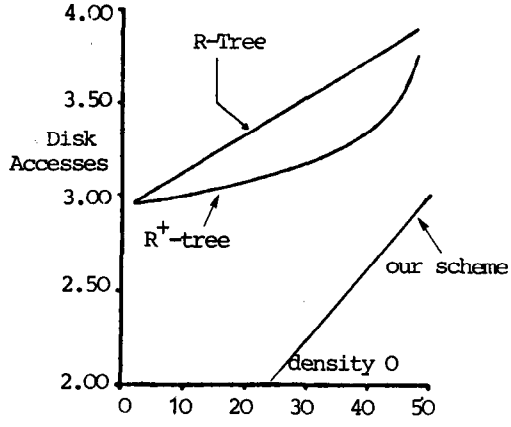

Figure 12: Disk accesses depending on the density 0 (b = 50, n = 100,000)

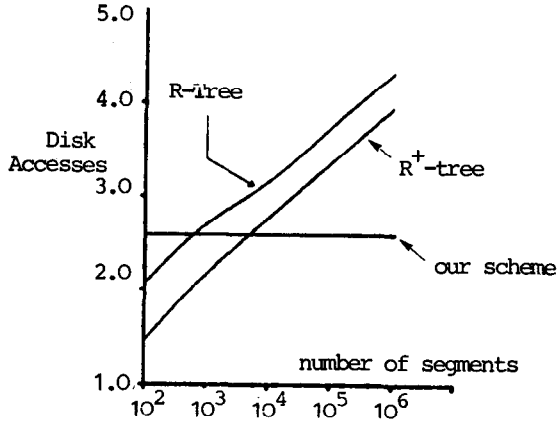

Figure 13: Disk accesses depending on the number of segments (b = 50, 0 = 40)