Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

409

A. Buchmann O. Günther
T.R. Smith Y.-F. Wang (Eds.)

Design and Implementation of Large Spatial Databases

First Symposium SSD '89
Santa Barbara, California, July 17/18, 1989
Proceedings

Springer-Verlag

Berlin Heidelberg New York London Paris Tokyo Hong Kong

Contents

Geographic Applications

Quadtrees

Modeling and Data Structures

Spatial Reasoning

Performance Comparison of Point and Spatial Access Methods *

Hans-Peter Kriegel, Michael Schiwietz
Ralf Schneider, Bernhard Seeger
Praktische Informatik, University of Bremen, D-2800 Bremen 33, West Germany

Abstract

In the past few years a large number of multidimensional point access methods, also called multiattribute index structures, has been suggested, all of them claiming good performance. Since no performance comparison of these structures under arbitrary (strongly correlated nonuniform, short "ugly") data distributions and under various types of queries has been performed, database researchers and designers were hesitant to use any of these new point access methods. As shown in a recent paper, such point access methods are not only important in traditional database applications. In new applications such as CAD/CIM and geographic or environmental information systems, access methods for spatial objects are needed. As recently shown such access methods are based on point access methods in terms of functionality and performance. Our performance comparison naturally consists of two parts. In part I we will compare multidimensional point access methods, whereas in part II spatial access methods for rectangles will be compared. In part I we present a survey and classification of existing point access methods. Then we carefully select the following four methods for implementation and performance comparison under seven different data files (distributions) and various types of queries: the 2-level grid file, the BANG file, the hB-tree and a new scheme, called the BUDDY hash tree. We were surprised to see one method to be the clear winner which was the BUDDY hash tree. It exhibits an at least 20 % better average performance than its competitors and is robust under ugly data and queries. In part II we compare spatial access methods for rectangles. After presenting a survey and classification of existing spatial access methods we carefully selected the following four methods for implementation and performance comparison under six different data files (distributions) and various types of queries: the R-tree, the BANG file, PLOP hashing and the BUDDY hash tree. The result presented two winners: the BANG file and the BUDDY hash tree. This comparison is a first step towards a standardized testbed or benchmark. We offer our data and query files to each designer of a new point or spatial access method such that he can run his implementation in our testbed.

Keywords : access methods, performance comparison, spatial database systems

---

* This work was supported by grant no. Kr 670/4-2 from the Deutsche Forschungsgemeinschaft (German Research Society) and by the Ministry of Environmental an Urban Planning of Bremen

1. Introduction

Access methods for secondary storage which allow efficient manipulation of large amounts of records are an essential part of a data base 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). A large number of multidimensional PAMs, also called multiattribute index structures, has been suggested in the past few years. Many of these PAMs claim to be "very efficient for arbitrary queries", to be "robust, coping well with arbitrary distributions", to "exhibit almost the same retrieval performance for independent nonuniform data distributions as for uniform distributions", or to "gracefully adapt to the actual data". However, no performance comparison of these structures under strongly correlated nonuniform data distributions and under various types of queries has been performed, simply because for many of these PAMs no implementations are available. In 1984 we have reported on a performance comparison of four PAMs, the grid file, two variants of multidimensional B-trees and the traditional inverted file, see [Kri 84]. However, all of these PAMs are outdated.

In this paper, we will present a performance comparison of the most promising PAMs under skewed data and under various types of queries. Our goal will eventually be to develop a standardized testbed or benchmark such that each designer of a new PAM may implement her or his method and run it against this benchmark. Such a performance comparison of PAMs will be the fundamentals of automatic physical database design tools that would choose a physical schema and then monitor the performance of the schema making changes as necessary.

Now, considering new applications such as Computer Aided Design/Computer Integrated Manufacturing (CAD/CIM), image processing and geographic or environmental information systems, PAMs are not sufficient. In particular, new access methods are necessary for the organization of multidimensional spatial objects, like rectangles, polygons 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.

The significance of efficient PAMs is underligned by the following facts. In [SK 88] we have shown that known SAMs for simple spatial objects (rectangles, intervals, etc.) are based on an underlying PAM using one of the following three techniques: clipping, overlapping regions and transformation. The better the underlying PAM, the better will be the performance of the resulting SAM. The distribution of objects which the underlying PAM handles is in almost all spatial applications nonuniform and strongly correlated; extremely correlated if the technique of transformation is used. As an underlying PAM we used in [SK 88] the most efficient multidimensional dynamic hashing scheme (MDH) without directory which is PLOP-Hashing [KS 88], mainly because it supports a nice adaption of the three different techniques. In this paper, we

will compare in part II the R-Tree, PLOP-Hashing, the BANG file and the BUDDY hash tree, all storing rectangles.

This paper is organized as follows. Part I deals with PAMs and consists of sections 2-5. In section 2 we will give a survey and classification of existing PAMs and we will justify our selection of PAMs for the performance comparison. In the third section we describe how we implemented the selected PAMs and we specify the general experimental setup for our comparisons. The result of the experiments are reported in section 4. In the following section 5 those results are interpreted. Furthermore, from the attempt to explain bad performance of the different PAMs, suggestions for improvements for most PAMs are made. Part II compares SAMs for rectangles and covers sections 6-8. In section 6 a brief survey and classification of existing SAMs for rectangles is presented. In the following section 7, we describe our general experimental setup and the selected SAMs. The results of the experiments are then reported in section 8. Section 9 concludes the paper.

Part I: Performance comparison of multidimensional point access methods (PAMs)

2. Classification and selection of PAMs

Even for someone working in this area, it is difficult to keep track of all multidimensional PAMs suggested until today. Most important for the performance of a multidimensional PAM under arbitrary (nonuniform correlated) data is the partitioning process, how the PAM adapts to the particular data distribution. Therefore, we will present a classification of existing multidimensional PAMs according to the way they partition the d-dimensional data space D. In the following classification we will not consider PAMs based on binary trees, such as kd-trees, since they are not suitable for the organization of data in secondary storage. Furthermore, we will omit variants of multidimensional B-trees [Kri 84] from our classification, because they cluster data according to a lexicographical ordering, instead of according to proximity in data space.

The basic principle of all multidimensional PAMs is to partition the data space into page regions, shortly regions, such that all records in one region are stored in one and the same data page. We will classify according to the following three properties of regions: the regions are pairwise disjoint or not, the regions are rectangular or not and the partition into regions is complete or not, i.e. the union of all regions spans the complete data space or not. Obviously, this classification yields six classes, four of which are filled with known PAMs.

class	property			PAM
	rectangular	complete	disjoint	
(C1)	X	X	X	interpolation hashing [Bur 83], MOLHPE [KS 86], quantile hashing [KS 87], PLOP-hashing [KS 88], k-d-B tree [Rob 81], multidimensional extendible hashing [Tam 82,Oto 84], balanced multidimensional extendible hash tree [Oto 86], grid file [NHS 84], 2-level grid file [Hin 85], interpolation-based grid file [Ouk 85]
(C2)	X	X		twin grid file [HSW 88]
(C3)	X		X	buddy hash tree [SFK 89], multilevel grid file [WK 85]
(C4)		X	X	B^+-tree with z-order [OM 84], BANG file [Fre 87], hB-tree [LS 89]

Table 1 : Classification of multidimensional PAMs.

As mentioned before our goal is to find PAMs with a good overall performance under nonuniform correlated data. Since it was not feasible to implement and compare all of the structures in the above classification, we selected the following 4 PAMs for implementation and comparison: the 2-level grid file, the BANG file, the hB-tree and the buddy hash tree. Before describing the selected PAMs in more detail, we will justify why we restricted our comparison to these four structures.

Considering class C 1, the most promising structures definitely are the interpolation-based grid file and the balanced multidimensional extendible hash tree. However, both structures can be obtained as a special case of the buddy hash tree by restricting the properties of the regions. Therefore these two PAMs need not to be implemented. We do not include the best multidimensional dynamic hashing scheme without directory, PLOP hashing, since it is efficient only for weakly correlated data, but not for strongly correlated data. From class C 1 we selected the 2-level grid file because it is generally accepted to be "the measuring stick" and because its efficient Modula-2 implementation by Klaus Hinrichs [Hin 85] was available to us which we thankfully acknowledge.

From class C 4 we omitted the B^+-tree storing z-values from our comparison, because both implemented PAMs, the BANG file and the hB-tree are improvements of the basic B^+-tree storing z-values. We decided to implement the buddy hash tree (class C 3) due to its non-complete partition of the data space thus avoiding to partition empty data space. Since the concept of the twin grid file (class C 2) of organizing two dependent grid files at the same time is generally applicable to any PAM, we did not include it in our comparison. It might be worth investigating the application of this principle to the winners of our comparison.

In the following, we will present a short description of the selected PAMs. This description is

slightly longer in case of the latest PAM, the buddy hash tree, since its paper might not be readily available.

The 2-level grid file was first suggested in the original grid file publication [NHS 84] and then described in detail and implemented in [Hin 85]. The basic idea is to manage the grid directory with another grid file. This 1st level grid directory is a scaled-down version of the original grid directory in which the limit of resolution is significantly coarser. Since the 2nd level grid files are independent from each other, this 2-level approach supports a better adaption to nonuniform distributions than the original 1-level grid file. However, the 1st level grid directory still grows superlinearly, just starting its superlinear growth later. Let us emphasize that the regions in the 2-level grid file are rectangular.

In order to adapt to the clustering of points in the data space, Freeston has suggested the BANG file (Balanced and Nested Grid file) [Fre 87] using the concept of nested regions. As in the 2-level grid file the data space is partitioned by rectangular shaped basic regions. However, contrary to the 2-level grid file, regions may be formed from these basic regions using the difference operation. The difference operation is applied to nondisjoint basic regions where one of them completely contains the others. Thus this operation supports a process of nesting which produces non-rectangular shaped logical regions. This process of nesting is applied to data pages and equivalently to directory pages. Obviously the motivation of the BANG file was a graceful adaption to object distributions where almost all of the data occurs in a few relatively small cluster points.

Conceptually similar to the BANG file, the hB-tree (holey brick tree) [LS 89] allows non-rectangular shaped regions on the level of data pages and more important on the level of directory pages. Contrary to the BANG file, such a region is generated by union of rectangular shaped basic regions. This potentially more efficient constructive method (versus the descriptive method in the BANG file), however, trades in again one of the basic disadvantages of the 2-level grid file: a logical region may need more than one directory entry.

Both, the BANG file and the hB-tree use a balanced search tree structure for the directory. The BANG file directory organizes a hash-based partition of the data space, whereas the hB-tree uses a kd-tree-type node organization in the directory, to reflect a median-based partitioning. Thus the BANG file is a hashing scheme with a tree-structured directory, hash tree for short, organizing the embedding data space, whereas the hB-tree is a search tree, organizing the specific set of data. To be precise, the hB-tree is actually a search graph due to its duplicate directory entries.

For none of the two structures a deletion algorithm has been specified. From our experience having implemented both of them, we believe that an efficient deletion algorithm will be especially hard to design for the hB-tree.

All existing PAMs including the 2-level grid file, the BANG file and the hB-tree have the following property in common: they partition the complete data space. More exactly, the union of all partitioning blocks spans the complete data space. Consequently empty data space is partitioned,

even if it is partitioned efficiently as in the case of the BANG file.

The goal of the buddy hash tree [SFK 89] is not to partition empty data space at all, even more, to partition the data space into nearly minimal bounding rectangles of objects. As the name says, it is, similar to the BANG file, a dynamic hashing scheme with a tree-structured directory where the leaves of the directory point to the data pages. A (page) node of the directory contains a list of entries (R, P) where $R \subseteq D$ is a d-dimensional rectangle in the data space D and P is a pointer to a subtree containing all points (records) in R. R is the minimal bounding rectangle of the points and subrectangles obtained by recursive halfing of the data space. The partitioning hyperplanes are parallel to the axis of the data space.

Consider an entry (R, P) in a directory node where P refers to a son $((S_1, P_1), ..., (S_k, P_k))$, $k \geq 1$. Then the following two conditions are fulfilled:

(i) $S_i \cap S_j = \emptyset \quad \forall_{i,j} \in \{1, ... k\}, i \neq j$

(ii) $\bigcup_{i=1}^{k} S_i \subseteq R$

Condition (ii) implies the important property of the buddy hash tree that it does not have to partition the complete data space. Together with the concept of minimal bounding rectangles condition (ii) implies that empty data space is not partitioned at all. Conditions (i) and (ii) have already been incorporated in the multilevel grid file [WK 85]. However, additionally to the multilevel grid file the buddy hashtree exhibits the following performance improving properties:

(1) Each directory node contains at least two entries.

(2) An overfilled page (data page or directory page) is always split in a minimal way i.e. the "minimal bounding rectangle property" is not destroyed by page splitting.

(3) Except for the root of the directory, there is exactly one pointer referring to each directory page.

(4) Let (R, P) be an entry in a leaf of the directory, i.e. P points to a data page. Then there may exist other pointers $P_1, ... P_k$, and accordingly directory entries $(R_1, P_1), ... (R_k, P_k)$, $k \geq 1$, iff

   (a) the rectangle R contains less than b/2 records (points), where b is the capacity of a directory page

   (b) the entries $(R_i, P_i), 1 \leq i \leq k$, are accomodated in the same leaf of the directory as (R, P).

The balanced multidimensional extendible hash tree and the multilevel grid file are artificially balanced by allowing one entry in a directory page. Due to property (1) the buddy hash tree shortens paths by omitting directory pages with one entry. Thus the buddy hash tree is not balanced, i.e. the leaves of the directory may be on different levels of the tree. We would like to emphasize that this is a performance improvement for all operations (queries and updates) compared to the balanced

competitors of the buddy hash tree. An important performance measure for a tree-structured directory is the maximum height of the directory. The maximum height h_{max} of the buddy hash tree is:

$$h_{max} \leq \log_{b/2}\left(\frac{n}{b/2} - w\right) + \frac{w - \log_2 b}{b}$$, where : n is the number of records

b is the capacity of a directory page

$|D| = 2^w$

Obviously , $\left(\frac{n}{b/2} - w\right) \ll n$ and $\frac{(w - \log_2 b)}{b} < 2$ is fulfilled for most applications.

Property (2) guarantees that for answering queries no pages are accessed and searched which do not contain an answer. Properties (1) and (3) imply that the directory grows linearly in the number of records under all circumstances. Property (4) results in a high storage utilization. However, the most important of these properties is property (2), the minimal bounding rectangle property which avoids partitioning empty data space.

Implementation specific details as well as the general experimental setup for our comparisons are described in the next section.

3. Experimental setup

We ran the performance comparisons on SUN workstations (3/50 and 3/60) under UNIX using Modula-2 implementations of the selected PAMs. We will first describe in more detail how we implemented these PAMs.

As mentioned before, there exists an efficient fine-tuned and well-tested Modula-2 implementation of the 2-level grid file [Hin 85], in the following tables and figures abbreviated by GRID. We are thankful to Klaus Hinrichs for making this implementation available to us. Since we use GRID as a measuring stick for the other PAMs, we will standardize the number of page accesses for range queries and partial match queries of GRID to 100 % for the sake of an easier comparability.

Contrary to the 2-level grid file, the BANG file implementation is not publicly available from the ECRC, Munich, West Germany (European Computer-Industry Research Centre). Thus we had to implement the BANG file, in the following comparisons abbreviated by BANG, on our own. In [Fre 87] the search path in an exact match query may be longer than the height of the tree. This results in a performance penalty particularly for range queries with small volume. This phenomenon is caused by the fact that the original BANG file suggestion does not fulfill the so-called "spanning property" which requires each directory node and thus each region to be completely spanned by its entries. Our implementation is according to the original BANG file concept [Fre 87] and does not yet include the spanning property. We are presently incorporating this spanning property in our

implementation and we will investigate the potential improvement. Furthermore, we are presently extending our implementation from fixed-length to variable-length directory entries which is incorporated in BANG*.

When we decided in September 1988 to include the hB-tree in our comparison of PAMs, no implementation was available. Our implementation of the hB-tree, denoted HB in the following figures and tables, gracefully follows the specification in [LS 89]. Additionally, we have implemented an optimized choice of the split axis which minimizes the margins of the regions in order to improve range query performance.

Obviously we had to implement the buddy hash tree [SFK 89] on our own. The implementation of the directory is very general, i.e. it is prepared to support a neighbor system. Since we decided for a special case of the neighbor system, the buddy system, there is room for improvement in the directory implementation which may easily result in an increase of the average branching factor of at least 40 %. To be fully dynamic we have incorporated a deadlock algorithm which contrary to the 2-level grid file is not a "must" in the buddy hash tree. Underfilled regions of highly varying sizes may not be merged in the original buddy hash tree because only rectangular regions are permissible. Thus it is possible to pack (merge) data pages such that the pointers to those data pages originate from one and the same directory page. For the sake of avoiding an unlimited number of indirect splits we have restricted "packing" to data pages. We have implemented the unpacked version, abbreviated by BUDDY in the comparison, and we have generated the packed version, called BUDDY+, by computation and simulation from the BUDDY implementation.

In order to compare the performance of the PAMs, we generated seven 2-dimensional datafiles (F1) - (F7) where (F1) - (F6) consists of 100 000 records without duplicates. (F7) consists of real cartography data and actually contains 81 549 records without duplicates. We consider records whose keys are in the unitcube $[0,1)^d$, since some of the PAMs require this. In the following, N(m,v) denotes a Gaussian distribution with mean value m and variance v. Below we will give a specification of the data files (F1) - (F7) which are additionally depicted in figure 3.1:

(F1) **"Diagonal"** :
The records follow a uniform distribution on the main diagonal.

(F2) **"Sinus Distribution"** :
The records follow a sinus curve, more precisely the x-values are uniformly distributed and the y-values follow a Gaussian distribution with mean value sin(x) and variance 0.1.

(F3) **"Bit Distribution"** :
The records follow a bit distribution bit(z) with parameter z, $0 \leq z \leq 1$. Each key component K can be represented as a bitstring $(b_1, b_2, \ldots)$, where

$$K = \sum_{j \geq 1} b_j \cdot 2^{-j}$$

The key component K follows a bit distribution bit(z) with parameter z, if for any j, the bit b_j satisfies $Pb(b_j = 1) = z$, where Pb(X) denotes the probability that event X is true. For our testfile we have chosen z = 0.15.

(F4) **"x-Parallel"** :
the x-values are uniformly distributed and the y-values follow an N(0.5,0.01) distribution.

(F5) **"Cluster Points"** :
The records follow a 2-dimensional independent Gaussian distribution with variance 0.05 (in x- and y-direction) around the centers of the cluster points as mean values and the records are inserted finishing one cluster point before starting the next.

(F6) **"Uniform Distribution"** :
The records follow a 2-dimensional independent uniform distribution. Since there is no need, this distribution is not depicted in figure 3.1.

(F7) **"Real Data"** :
Consists of real cartography data representing the elevation lines in a "rolling-hill-type" area in the Sauerland, West Germany. The points are obtained as interpolation points of the elevation lines. Since the data is originally stored in a quad-tree, it is inserted in a sorted sequence which is due to the partitioning sequence of the quad-tree. We thankfully acknowledge receiving this data from the Landesvermessungsamt NRW, Bonn, West Germany.

For each of the files (F1) - (F7) we generated the following five query files for comparing the selected PAMs:

(RQ1 20 quadratic range queries with volume 0.1 %, where the center of the square follows a uniform distribution.

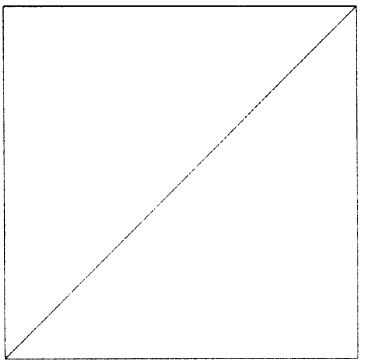
(RQ2 20 quadratic range queries with volume 1 %, where the center of the square follows a uniform distribution.

(RQ3 20 quadratic range queries with volume 10 %, where the center of the square follows a uniform distribution.

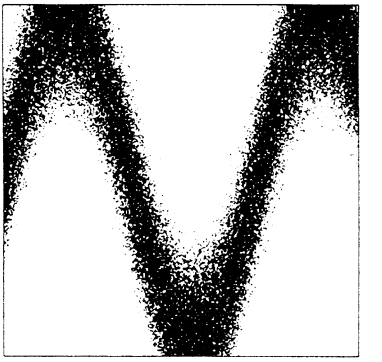
(PMQ1) 20 partial match queries where the specified x-value is uniformly distributed and the y-value is unspecified.

(PMQ2) 20 partial match queries where the specified y-value is uniformly distributed and the x-value is unspecified.

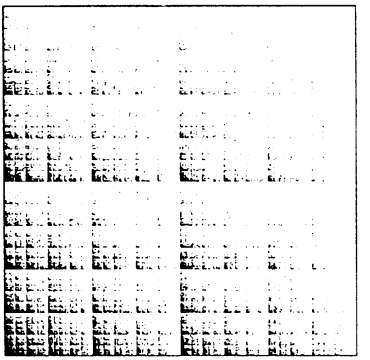
Here the volume of a range query is the volume of the specified range divided by the volume of the data space. For these queries we have computed the average number of disk accesses per query where the average is taken over 20 queries.



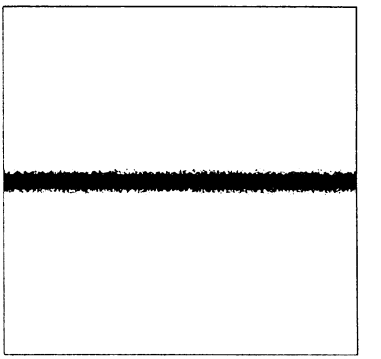
(F1) Diagonal



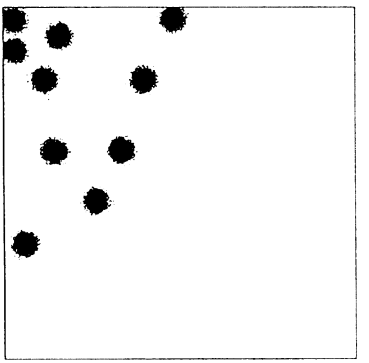
(F2) Sinus Distribution



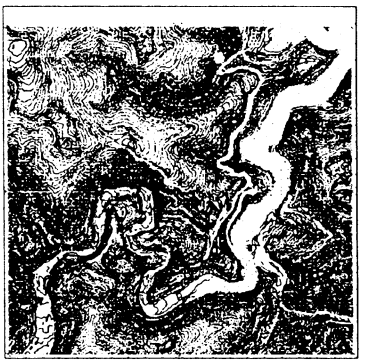
(F3) Bit Distribution



(F4) x - Parallel



(F5) Cluster Points



(F6) Real Data

Fig. 3.1 : Data - Distributions

As mentioned before, for the BANG-file and the hB-tree no deletion algorithms have been specified. Therefore, for our comparison we only consider the case of the growing file.

In order to keep the performance comparison manageable (we already had more than 2.7 billion insertions), we have chosen the page size for data pages and directory pages to be 512 bytes which is at the lower end of realistic page sizes. Using small page sizes, we obtain similar performance results as for much larger file sizes, e.g. a doubling of the page size can accomodate an eight times higher file size within the same directory height for tree-based directories (BANG, HB, BUDDY). We want to emphasize that for the 2-level grid file the 1st level grid directory is always kept in main memory whereas for the other methods with their tree-based directories only the root page is main memory resident. Since the 1st level grid directory grows superlinearly, this may become infeasible (e.g. we had to keep up to 45 directory pages in main memory for only 100 000 records). Furthermore, in order to support update operations, in tree-based directories we additionally store the last accessed search path in a buffer and analogously for the 2-level grid file the last two accessed pages. Naturally this buffer for the search path is dynamically growing and shrinking according to the height of the tree.

Summarizing we can state that the performance results in the next section of BANG, HB and BUDDY hold as well for much larger file sizes whereas GRID will perform worse for larger file sizes due to its superlinear growth of the 1st level directory for nonuniform distributions.

4. Results of the experiments

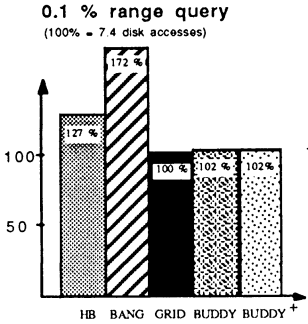
As mentioned before, for the query types (RQ1) - (RQ3) and (PMQ1), (PMQ2) we will report the average number of disk accesses per query in the following tables. For the sake of an easier comparability, we have standardized the average number of page accesses for these queries in GRID to 100 %. Under the considerations of real-life applications and robustness, we have further visualized our results for the datafiles "Real Data", "Cluster" and "Diagonal".

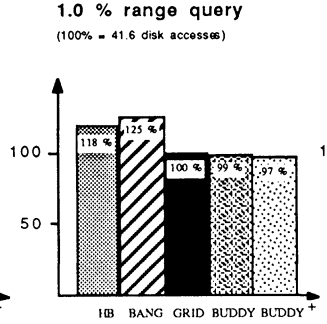
During and after building up each datafile from empty, the following parameters were measured:

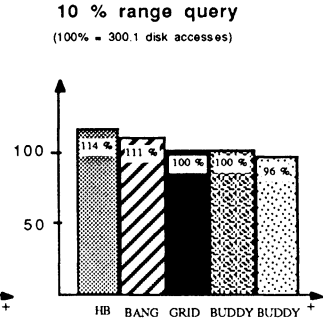
1. the storage utilization, denoted by stor.
2. the ratio of directory pages to data pages, denoted by dir/data.
3. the average number of disk accesses for an insertion (read and write) averaged over all 100000 or 81 549 insertions, denoted by insert.
4. the height of the directory after completely building up the file, denoted by h.

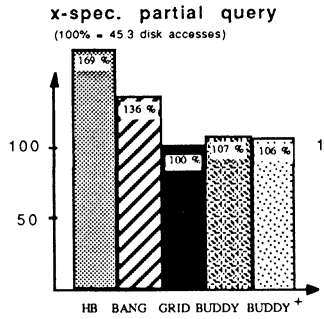
The results of the experiments are reported in the following figures and tables:

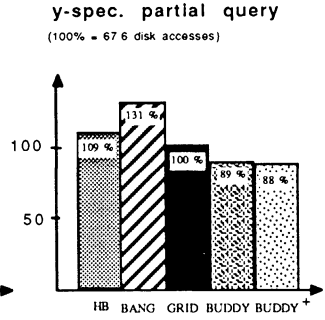
Real Data





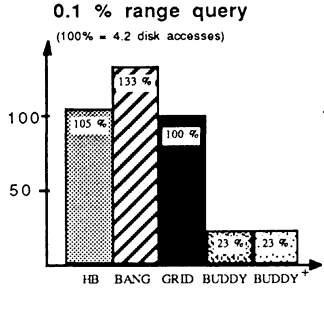


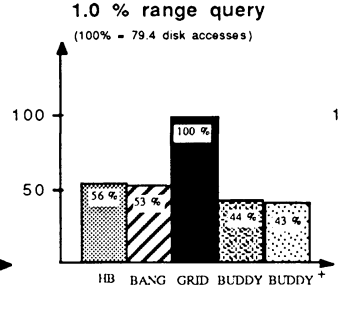


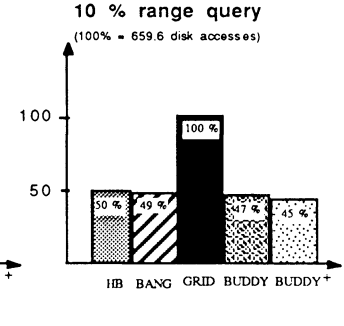


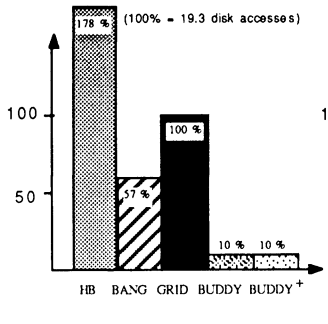
	stor	dir/data	insert	h
HB	62.9	4.20	0.39	3
BANG	64.1	2.44	0.38	2
GRID	67.2	1.36	0.41	2
BUDDY	67.9	1.79	0.38	3
BUDDY+	72.4	1.91	—	3

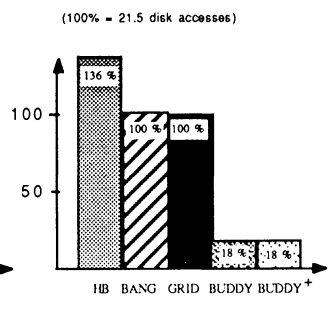
Diagonal





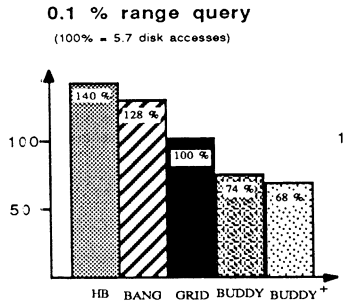


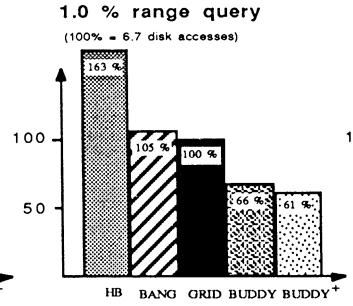


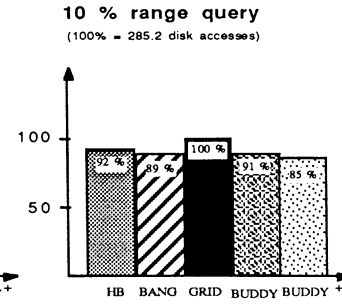


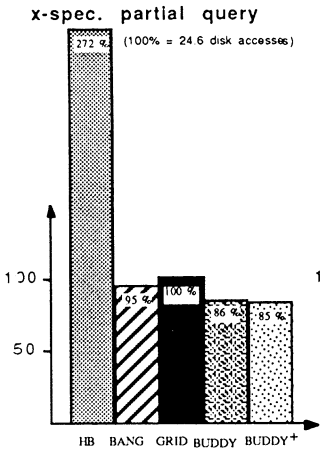
	stor	dir/data	insert	h
HB	70.1	3.61	3.29	3
BANG	69.9	2.34	3.03	3
GRID	35.4	8.98	3.13	2
BUDDY	69.9	2.27	3.19	2
BUDDY+	74.1	2.41	—	2

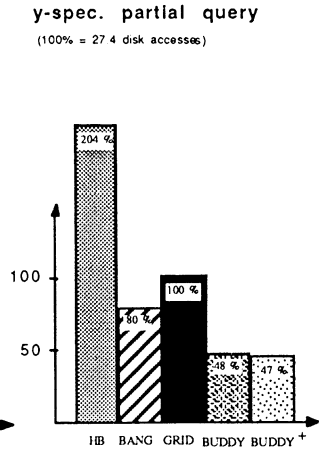
Cluster Points











	stor	dir/data	insert	h
HB	69.2	3.88	2.78	3
BANG	68.8	2.30	2.56	3
GRID	62.1	2.24	2.44	2
BUDDY	67.1	4.00	2.66	3
BUDDY+	71.5	4.25	/	3

Uniform Distribution

	range query			partial query		stor	dir/data	insert	h
	0.1%	1.0%	10%	x-spec	y-spec				
HB	113.3	104.3	103.9	137.3	92.7	69.9	3.53	3.29	3
BANG	113.9	105.8	101.9	110.6	103.5	70.1	2.35	3.06	3
GRID	100.0	100.0	100.0	100.0	100.0	70.2	1.12	2.90	2
BUDDY	101.7	102.7	101.2	108.3	100.0	70.2	2.28	3.19	2
BUDDY+	101.2	100.5	96.8	107.4	99.6	74.5	2.42	/	2

Sinus Distribution

	range query			partial query		stor	dir/data	insert	h
	0.1%	1.0%	10%	x-spec.	y-spec.				
HB	105.4	103.4	100.2	121.2	97.5	69.1	3.77	3.29	3
BANG	139.2	109.5	100.1	111.9	107.3	69.6	2.33	2.95	3
GRID	100.0	100.0	100.0	100.0	100.0	68.2	1.67	2.97	2
BUDDY	97.1	98.4	98.3	92.2	91.9	68.8	2.10	3.21	2
BUDDY+	96.6	95.1	93.8	89.8	90.3	72.9	2.22		2

Bit Distribution

	range query			partial query		stor	dir/data	insert	h
	0.1%	1.0%	10%	x-spec.	y-spec.				
HB	77.1	61.2	59.2	52.7	50.8	69.5	3.72	3.28	3
BANG	145.0	84.3	64.0	44.8	64.5	67.3	2.42	2.96	3
GRID	100.0	100.0	100.0	100.0	100.0	42.4	2.75	3.03	2
BUDDY	115.6	105.6	99.2	48.4	69.7	43.0	5.10	3.62	3
BUDDY+	105.5	89.6	67.5	46.1	66.5	71.0	8.42		3

x-Parallel

	range query			partial query		stor	dir/data	insert	h
	0.1%	1.0%	10%	x-spec.	y-spec.				
HB	94.9	89.2	91.1	132.4	59.6	69.6	3.62	3.29	3
BANG	126.5	100.1	95.8	83.6	114.7	65.4	2.19	3.03	3
GRID	100.0	100.0	100.0	100.0	100.0	62.9	3.77	3.01	2
BUDDY	74.5	83.1	92.3	72.8	50.4	67.2	2.45	3.21	2
BUDDY+	72.4	78.5	87.3	72.6	50.0	71.1	2.60		2

5. Interpretation of the results

Obviously, a physical database designer or a user of a database system can select from the above distribution mix those distributions which are typical and representative in his application. He will then choose the winner in those typical distributions. As a decision support for someone aiming for robustness and good average performance we present the following table 5.1. As mentioned before, our BANG file implementation incorporates fixed-length directory entries. For curiosity and as originally intended, we have generated a variable-length version, called BANG*, by simulation from the BANG implementation. We will only present the averaged results of BANG* in the following two tables 5.1 and 5.2.

In table 5.1 for the parameters stor and insert we computed the unweighted average over all seven distributions (datafiles). As an indicator for the average query performance we present the parameter query average which is averaged (unweighted) over all five query types for each distribution and then averaged over all seven distributions. The goal of this indicator is to help make things more clear, at first glance; however, we are aware that such an average implies a loss of information. The loss of information is considerably less in table 5.2 where the parameter query is displayed for each distribution as an average over all five types of queries.

	query average	stor	Insert
HB	110.9	68.6	2.80
BANG	102.6	67.9	2.43
BANG*	95.8	67.9	2.49
GRID	100.0	58.3	2.56
BUDDY	80.2	64.9	2.78
BUDDY+	76.6	72.5	/

Table 5.1: unweighted average over all 7 distributions

	uniform	sinus	bit	x-par.	real data	diagonal	cluster
HB	110.3	105.5	60.2	93.4	127.4	105.0	174.2
BANG	107.1	113.6	80.5	104.1	135.0	78.4	99.4
BANG*	100.2	108.0	72.8	99.8	131.8	68.2	90.1
GRID	100.0	100.0	100.0	100.0	100.0	100.0	100.0
BUDDY	102.8	95.6	87.7	74.6	99.4	28.4	73.0
BUDDY+	101.1	93.1	75.0	72.2	97.6	27.8	69.2

Table 5.2 : unweighted average over all 5 types of queries depending on the distribution

In the following, we will discuss the performance of each PAM in the sequence in which they appear in the tables focussing on the average over the 5 types of queries.

Considering the indicator query average, HB would be the looser. However, this simple approach is not fair. For the bit distribution HB clearly outperforms its competitors and for the x-parallel HB closely follows BUDDY and BUDDY$^+$. This good performance for the bit distribution is a consequence of the median-based partitioning, whereas the performance for the x-parallel profits from the additional feature of the minimized margins of the regions which was not an ingredient in the original specification [LS 89], but was incorporated in our implementation. For all other distributions (Real Data, Diagonal, Cluster, Sinus Distribution and Uniform Distribution) the average over all 5 types of queries is clearly worse than the 100 % value of GRID. More specifically, for Cluster, Diagonal and Uniform Distribution HB is the extreme looser in average query performance with values up to 272 %. Thus HB does not guarantee robustness. Although HB is the only PAM incorporating the efficient median-partitioning, it suffers from the following severe disadvantages:

(i) the height of the directory is in most experiments one more than in the other PAMs.
(ii) considering the partitions of HB for all distributions we observe that HB often partitions empty data space with unnecessarily fine granularity.
(iii) the directory may contain duplicate entries in two respects:
  (a) the father of a directory node may contain subtrees of its sons
  (b) different directory entries may point to one and the same page (directory or data pages).

From the above it follows that the hB-tree is actually a graph. We believe that the only way to improve HB is to incorporate the concept of not partitioning empty data space. With this and the median partition it might become very competitive.

As mentioned before, the GRID implementation [Hin 85] always keeps the 1st level grid directory in main memory whereas for the other PAMs only the root page of the directory is main memory resident. Since it was crucial to change the GRID implementation to allowing only one root page of the directory in main memory, we accepted that the relative ranking of GRID, our 100 % measuring stick, is too good in comparison to the other structures. To clarify this: for the Diagonal Distribution the 1st level grid directory needed 45 directory pages in main memory, which is sufficient for BANG, BANG*, BUDDY and BUDDY$^+$ to keep the complete directory in main memory. Thus the rating of GRID in a comparable environment would be considerably worse. With the available implementation, GRID outperforms its competitors for uniform distribution as expected. If we exclude HB from our considerations it performs considerably worse than BANG and BUDDY for Diagonal, Bit Distribution and Cluster. Our comparisons show that GRID is not robust against arbitrary data.

Considering BANG and BANG* for the indicator query average, the concept of nested regions

seems not to imply any improvement over GRID. However, BANG and BANG* turn out to be more robust towards ugly distributions than HB and especially GRID are. Looking more closely at the different queries, we realize that BANG performs very poorly for small range queries. This is a direct consequence of the not incorporated "spanning property" and will be improved by its implementation. A further disadvantage in robustness of BANG is the fact that different sequences of insertions imply different partitions. In particular sorted insertions seem to result in low storage utilization and poor retrieval performance.

For distributions where large portions of empty data space occur, i.e. x-Parallel, Diagonal, Sinus Distribution and Cluster Points, BANG and BANG* perform considerably worse than BUDDY. Looking at the ingredients of both PAMs it follows that incorporating an adapted concept of minimizing regions into BANG will improve the retrieval performance to some extent.

However, a consequent minimization of regions will lead to an incomplete partition of the dataspace, i.e. not partitioning empty data space, and thus to the most performance-important ingredient of BUDDY.

Considering the indicator query average, BUDDY and BUDDY$^+$ offer themselves to be the winners of our comparison. It is interesting to observe that BUDDY does not fulfill the often cited rule "best storage utilization - best query performance". Even the improvement in storage utilization of BUDDY$^+$ over BUDDY is not adequately reflected in the improvement of the retrieval performance. As mentioned before, we have to take a closer look at the different distributions. The only distributions where BUDDY and BUDDY$^+$ are not the winners are the Uniform and the Bit Distributions, see table 5.2. According to [SFK 89], the Bit Distribution bit(z), $0 \leq z \leq 1$, is the worst case distribution for BUDDY and BUDDY$^+$ when z approaches 0. Even for its worst case distribution BUDDY$^+$ is 3rd winner for the average query performance. This underlines the robustness of the structure. By the way, the motivation for the design of BUDDY$^+$ to improve the storage utilization stems from exactly this pathological distribution. For Uniform Distribution BUDDY and BUDDY$^+$ are within a 3 % margin of GRID, the winner. This is surprising for a scheme designed for nonuniform data incorporating the complex structural concept of not partitioning empty data space.

In all distributions, with the exeption of the Uniform and Bit Distribution, BUDDY and BUDDY$^+$ are the clear winners in the average query performance and BUDDY$^+$ wins in the storage utilization with more than 71 %. BUDDY and BUDDY$^+$ clearly outperform their competitors if at least one of the following two data characteristics occur:

(C1) densely populated and unpopulated areas vary over the data space,

(C2) sorted data is inserted.

Sorted insertions frequently occur in real-life applications, either sorted by some local ordering such as clusters or quadrants or by lexicographical ordering.

Whereas other PAMs suffer from characteristics (C1) and/or (C2), BUDDY and BUDDY$^+$ behave robust, see distributions "Diagonal" and "Cluster Points".

Part II: Performance comparison of spatial access methods (SAMs)

6. Classification and selection of SAMs

Even for someone working in this area, it is difficult to keep track of all SAMs suggested until today, because every multidimensional PAM can easily be extended to a SAM using the techniques of clipping, overlapping regions and transformation.

In this section we will provide an overview of spatial access methods 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. 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. We do not consider more complex approximations of spatial objects such as the cell-tree [Gün 89] in this paper.

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 following we give a short describtion of the several techniques of extending PAMs to SAMs. The interest reader can find these techniques explained in more detail in [SK 88]

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

Overlapping regions

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 brief summmary of the concepts of 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 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.

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 heavily depends on the amount of overlap, as shown in [SFR 87].

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 $u_j \in [l_j,1)^2$, l_j=1,2.

However, the choice of the parameters can influence performance and characteristics of the SAM.

7. Experimental setup

We ran the performance comparisons on SUN workstations (3/50 and 3/60) under UNIX using Modula-2 implementations of the selected SAMs.

Not much has to be said with respect to the selection and the implementation of the SAMs. The measuring stick in our comparison is the R-tree. Our implementation of the R-tree gracefully follows the specification of the R-tree in [Gut 84]. According to Diane Greene's [Gre 89] implementation we chose at first a minimum storage utilaization of 50%, but our tests showed that the R-tree exhibits best retrieval performance for a minimum storage utilization of 30%. The "measuring stick role" of the R-tree is particulary justified because it basically wins the performance comparison by Diane Greene [Gre 89]. The obvious competitors are the two best PAMs in our comparison of PAMs, BUDDY and BANG.

Using the technique of transformation with corner representation we extended both our BANG and our BUDDY implementation to SAMs. To be precise, we used the BANG* implementation for rectangles, but for the sake of simplicity we will denote it by BANG. Which is the more efficient representation to use with transformation, the corner or the center representation? In order to answer this question Bernhard Seeger experimentally compared both representaions for BUDDY in his PhD thesis [See 89] for different types of queries and different distributions of rectangles. Simply speaking the corner representation yields approximately half the number of page accesses of the center representation. The basic reason is that for the corner representation the limits of the query ranges (areas) are parallel to the partitioning lines of BUDDY (and BANG) and thus the margin of the query range intersects fewer partitioning blocks than for the center representation. Now we have to make a statement with respect to the PAM versions of BUDDY and BANG on which we applied the corner representation. The BANG version was more refined than in our PAM comparison, already incorporating the spanning algorithm, whereas the BUDDY version was the first version, even without packing. Thus the results of BUDDY can easily be improved by incorporating packing and other refinements whereas BANG leaves practically no more room for improvement. In the final version of the paper we will have a refined version of BUDDY ready for our experiments.

The last SAM is based on PLOP-Hashing and uses the technique of overlapping regions as described in [SK 88] in detail.

In order to compare the performance of the SAMs, we generated five 2-dimensional datafiles (F1) - (F5) consisting of 100 000 rectangles without duplicates. A rectangle is characterized by its center and its x- and y-extension from the center. We consider rectangles which are in the unitcube $[0,1)^2$, since some of the SAMs require this. In the following, N(m,v) denotes a Gaussian distribution with mean value m and variance v. Below we will give a specification of the data files (F1) - (F5).

(F1) **"Uniformsmall-Distribution"** :

The centers of the rectangles follow a 2-dimensional independent uniform distribution within $[0.1)^2$. The extensions in x- and y- direction follow a uniform distribution in [0,0.005].

(F2) " **Uniformlarge-Distribution**" :
The centers of the rectangles follow a 2-dimensional independent uniform distribution within $[0.1)^2$.The extensions in x- and y- direction follow a uniform distribution in [0,0.5].

(F3) **"Gaussiansquare-Distribution"** :
The centers of the rectangles follow a 2-dimensional independent Gaussian distribution N(0.5,0.25) in x- and y- direction. The extensions in x- and y- direction follow a uniform distribution in [0,0.05].

(F4) " **Gaussianslim-Distribution**" :
The centers of the rectangles follow a 2-dimensional independent Gaussian distribution N(0.5,0.25) in x- and y-direction. The extension in x-direction follows a uniform distribution in [0,0.05] and the extension in y-direction follows a uniform distribution in [0,0.25].

(F5) **"Diagonal-Distribution"** :
First we generated two dimensional points which follow a uniform distribution on the main diagonal. Then the x- and y-coordinate of these points follow a Gaussian distribution N(0,0.5) .The two dimensional points generated in this way are the centers of the rectangles. The extensions in x- and y- direction follow a uniform distribution in [0,0.2].

For each of the files (F1) - (F5) we generated queries of the following four types:

"rectangle containment":
Given a d-dim. rectangle $S \subseteq E^d$, find all d-dim. rectangles R in the file with $R \subseteq S$.

"rectangle enclosure":
Given a d-dim. rectangle $S \subseteq E^d$, find all d-dim. rectangles R in the file with $R \supseteq S$.

"rectangle intersection":
Given a d-dim. rectangle $S \subseteq E^d$, find all d-dim. rectangles R in the file with $S \cap R \neq \emptyset$.

"point query":
Given a d-dim. point $P \in E^d$, find all d-dim. rectangles R in the file with $P \in R$.

For each of the files (F1) - (F5) we performed 500 queries for each SAM. By definition, each of the query types rectangle intersection, rectangle enclosure and rectangle containment uses a query rectangle. Therefore we generated 160 query rectangles with uniformly distributed centers for each of the three query types. In order to analyze the influence of the query rectangles on the performance, we are varying their size and shape. We generate 20 "square shaped" rectangles of sizes 0.1% 0.5%, 1% and 5% where the length of the rectangles is uniformly distributed between

1/2 squareroot(size) and 3/2 squareroot(size). Analogously, we generate 20 "slim" rectangles of sizes 0.1%, 0.5%, 1% and 5% where the length of the rectangles is uniformly distributed between 1/10 squareroot(size) and 19/10 squareroot(size). With these 160 query rectangles we perform the three query types rectangle intersection, rectangle enclosure and rectangle containment, thus yielding 480 queries. The remaining 20 queries are point queries, where the points follow a two dimensional independent uniform distribution.

8. Results of the experiments

As mentioned before, we will report in the following five tables the average number of disk accesses per query for each of the five files (F1) - (F5) and the four different query types.

Gaussianslim-Distribution

	point query	intersection	enclosure	containment
R-Tree	189.4	472.0	34.8	472.0
BANG	167.7	401.4	41.7	37.1
BUDDY	159.8	394.9	30.4	34.5
PLOP	273.6	637.3	55.5	637.3

Uniformsmall-Distribution

	point query	intersection	enclosure	containment
R-Tree	55.9	195.8	15.0	195.8
BANG	52.5	177.1	17.4	61.1
BUDDY	37.0	162.8	7.2	58.5
PLOP	41.4	172.9	6.1	172.9

Gaussiansquare-Distribution

	point query	intersection	enclosure	containment
R-Tree	86.5	266.7	14.0	266.7
BANG	68.8	236.3	16.0	68,2
BUDDY	57.6	232.6	6.4	65.7
PLOP	97.2	299.2	6.8	299.2

Uniformlarge-Distribution

	point query	intersection	enclosure	containment
R-Tree	742.8	988.2	518.7	988.2
BANG	388.6	603.8	239.4	20.2
BUDDY	380.2	593.3	231.2	18.0
PLOP	783.6	965.4	613.0	965.4

Diagonal-Distribution

	point query	intersection	enclosure	containment
R-Tree	283.4	568.2	163.7	568.2
BANG	187.8	413.3	97.2	25.6
BUDDY	187.5	421.0	92.9	22.9
PLOP	435.2	748.1	245.5	748.1

Similar as in our PAM comparison we computed the unweighted average over all five files and depict in the following table. In order to prevent overweighting of distributions with high number of page accesses, such as the Uniformlarge Distribution, we normalized the distributions by replacing the absolute values by percentage values where we use the R-tree as a 100% measuring stick. Additionally the average storage utilization denoted by stor and the average number of disk accesses for an insertion (read and write), averaged over all 100000 insertions when building up the file are presented in the following table.

	point query	intersection	enclosure	containment	stor	insert
R-Tree	100.0	100.0	100.0	100.0	67.6	110.3
BANG	76.1	79.5	91.2	14.3	68.5	2.88
BUDDY	66.9	77.6	56.5	13.5	65.5	2.92
PLOP	98.1	113.0	103.4	113.0	61.0	2.74

After running the experiments for the five files (F1) - (F5) and the four different query types we became aware that the performance comparison for rectangles is far more complex than the comparison for points for the following reasons:

1. The objects, here rectangles, are more complex than points. Whereas points as zero-size objects are determined by their position in dataspace, rectangles are determined by the following parameters: position, size, shape (square or long and slim) and degree of overlap. Obviously all of these parameters have to be extensively varied in a comparison.
2. The queries are more complex. One reason is that already the query object which is a rectangle in rectangle containment, rectangle enclosure and rectangle intersection is more complex. Furthermore, there are additional important operations and queries such as spatial join ("overlay two maps") and near neighbor-type queries.
3. The access methods are more complex. A SAM for rectangles is based on a PAM and uses one of the techniques clipping, overlapping regions and transformation. As shown in [SK 88] a hybrid method combining two techniques and avoiding their weak points improves performance over just using one of the techniques. Questions arise like which technique is best for which query type? For example, in our experiments it turned out that the technique of transformation was always best for the rectangle containment query. An additional example for the higher complexity of the access methods is the R-tree. Guttman's original design of the R-tree [Gut 84] can easily be improved by improving its split condition, e.g. by using Diane Greene's split condition [Gre 89]. Even this split condition can still considerably be improved as our implementations of Guttman's, Greene's and our own split conditions show.

From the above reasons it is obvious, that a considerably more extensive comparison for SAMs storing rectangles has to be performed. The presently available results indicate that BANG and particularly BUDDY are first choices.

9. Conclusions

In our performance comparison of point access methods, we were surprised to see one point method to be the clear winner. We had expected a much more complex result depending on the particular data distribution and on the particular query type. Summarizing the outcome of our comparisons we can state that the BUDDY hash tree exhibits an at least 20 % better average query

performance than its competitors and, even more important, is more robust under ugly data and queries. Looking at the results of the experiments and at the partitions of the data space more closely, it turns out that the good performance of the BUDDY hash tree is not by chance, but is due to the concept of not partitioning the complete data space. Thus it might be worthwhile to incorporate this performance improving concept into other methods, in particular into the BANG file.

From our comparison of spatial access methods for rectangles it follows that this comparison has to be performed with a considerably higher variation of object parameters (position, size, shape and degree of overlap), query parameters and techniques (clipping, overlapping regions and transformation). The presently available results indicate that BANG and particularly BUDDY both using transformation are first choices for spatial access methods storing rectangles.

Further work in this area should deal with performance comparisons of access methods for more complex spatial objects, such as polygons, where only very few access methods are known. Therefore access methods for complex spatial objects have to be designed and compared with the most promising candidate, the cell-tree [Gün 89].

As mentioned before this comparison is a first step towards a standardized testbed or benchmark. We offer our data and query files to everybody who wants to run his implementation in our testbed. At the same time, we are thankful for "hard" datafiles, in particular for "hard" real data.

Acknowledgement:

First of all, we would like to thank our colleagues Peter Heep and Stephan Heep for their valuable advice and support. Bernhard Seeger implemented the buddy hash tree, the implementation of the hB-tree is due to Michael Schiwietz and the BANG file was partially implemented by a group of computer science students and partially by Michael Schiwietz. Furthermore we are thankful to the Landesvermessungsamt NRW, Bonn West Germany, for making real cartography data available to us. Last, not least, we thank Ursula Behrend for professionally writing this manuscript.

References:

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

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

[Fre 87] M. Freeston: 'The BANG file: a new kind of grid file', Proc. ACM SIGMOD Int. Conf. on Management of Data, 260-269, 1987

[Gre 89] D. Greene: 'An Implementation and Performance Analysis of Spatial Data Access Methods', Proc. 5th Int. Conf. on Data Engineering, 606-615, 1989

[Gün 89] O. Günther: The design of the cell tree: An object-oriented index structure for geometric databases, in Proc. Fifth Intl. Conf. on Data Engineering, Feb. 6-10, 1989, Los Angeles

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

[Hin 85] K. Hinrichs: 'The grid file system: implementation and case studies for applications', Dissertation No. 7734, Eidgenössische Technische Hochschule (ETH), Zuerich, 1985

[HSW 88] A. Hutflesz, H.-W. Six, P. Widmayer: 'Twin grid files : space optimizing access schemes', Proc. ACM SIGMOD Int. Conf. on Management of Data, 183-190, 1988

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

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

[KS 87] H.P. Kriegel, B. Seeger: 'Multidimensional quantile hashing is very efficient for non-uniform distributions', Proc. 3rd Int. Conf. on Data Engineering, 10-17, 1987, extended version will appear in Information Science

[KS 88] H.P. Kriegel, B. Seeger: 'PLOP-Hashing: a grid file without directory', Proc. 4th Int. Conf. on Data Engineering, 369-376, 1988

[LS 89] D.B. Lomet, B. Salzberg: The hB-tree: A robust multiattribute search structure, in Proc. of the Fifth Int. Conf. on Data Engineering, Feb. 6-10, 1989, Los Angeles, also available as Technical Report TR-87-05, School of Information Technology, Wang Institute of Graduate Studies.

[NHS 84] J. Nievergelt, H. Hinterberger, K.C. Sevcik: 'The grid file: an adaptable, symmetric multikey file structure', ACM Trans. on Database Systems, Vol. 9, 1, 38-71, 1984

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

[OM 84] J.A. Orenstein, T.H. Merrett: 'A class of data structures for associative searching', Proc 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 181-190, 1984

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

[Oto 86] E. J. Otoo, : 'Balanced multidimensional extendible hash tree', Proc. 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 110-113, 1986

[Ouk 85] M. Ouksel: 'The interpolation based grid file', Proc. 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1985

[Rob 81] J. T. Robinson: '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

[See 89] Seeger, B.: 'Design and implementation of multidimensional access methods' in German), PhD thesis, Department of Computer Science, University of Bremen.

[SFK 89] B. Seeger, S. Frank, H.P. Kriegel: The buddy hash tree, English version in preparation, German version available as a Technical Report

[SFR 87] Sellis, T., Roussopoulos, N., Faloutsos, C.: 'The R^+-tree: a dynamic index for multi-dimensional objects', Proc. 13th Int. Conf. on Engeneering, 1988.

[SK 88] B. Seeger, H. P. Kriegel: 'Design and implementation of spatial access methods', Proc. 14th Int. Conf. on Very Large Databases, 360-371, 1988

[Tam 82] M. Tamminen: 'The extendible cell method for closest point problems', BIT 22, 27-41, 1982

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