Towards an Analysis of Range Query Performance in Spatial Data Structures*

Bernd-Uwe Pagel** Hans-Werner Six** Heinrich Toben** Peter Widmayer§

Abstract

In this paper, we motivate four different user defined window query classes and derive a probabilistic model for each of them. For each model, we characterize the efficiency of spatial data structures in terms of the expected number of data bucket accesses needed to perform a window query. Our analytical approach exhibits the performance phenomena independent of data structure and implementation details and whether the objects are points or non-point objects.

1 Introduction

In recent years, various efficient data structures for maintaining large sets of multidimensional geometric objects have been developed. Most of these structures have been designed for multidimensional points (see e.g. [2, 5, 7]). In typical applications, however, objects are arbitrarily geometric, i.e. non-point objects. In many situations, it has been proven to be useful to characterize non-point objects by their bounding boxes, i.e. minimal enclosing multidimensional intervals, serving as simple geometric keys. Hence, non-point data structures deal with multidimensional intervals (see e.g. [4, 5, 6]). Only the cell tree [3] does not use this approximation.

All proposals claim to improve the performance of spatial accesses and provide performance evaluations for range queries, which are the most popular spatial access operation. However, up to now, all evaluations have been carried out by simulations using the only assumption that ranges are windows of certain areas (e.g. 1%, 0.1% and 0.01% relatively to the area of the data space) and that window centers are uniformly distributed.

In this paper, we try to get one step further towards an understanding of window query performance of spatial data structures. We sketch four different classes of user-defined window queries and motivate their practical relevance. For each class, we derive a probabilistic model, which we use as basis for our analytical investigations. For each window query model, we derive a performance measure which characterizes arbitrary data space organizations in terms of the expected number of bucket accesses needed to perform a window query. Since it is well-known that in practical applications data bucket accesses exceed by far external accesses to the paged parts of the corresponding directory concerning frequency and execution time, the data space organization of a spatial data structure is essential for the window query performance. Hence, although our considerations are restricted to data bucket accesses, the real situation is still sufficiently reflected.

In contrast to purely experimental investigations which present the corresponding performance phenomena for specific implementations, specific object sets and specific window query patterns, our analytical approach exhibits the effects on a conceptual level, independent of data structure and implementation details, and even independent of whether the objects are points or non-point objects. We claim that our analysis sheds some new light on the complex spatial data structure field.

This paper is organized as follows. The next section reviews the basic concepts of spatial data management and introduces four different classes of user-driven window queries. In section 3, we provide a more formal look at the query classes by deriving a probabilistic model for each of them. In section 4, for each query model and arbitrary data space organizations, we present an analytical evaluation of the window

*This work has been supported by the Deutsche Forschungsgemeinschaft DFG, and by the ESPRIT II Basic Research Actions Program of the European Community under contract No. 6881 (AMUSING).

**FernUniversität Hagen, D-5800 Hagen

§ETH Zürich, CH-8092 Zürich

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

ACM-PODS-5/93/Washington, D.C.

© 1993 ACM 0-89791-593-3/93/0005/0214...$1.50

query performance in terms of the expected number of bucket accesses needed to perform a window query. Section 5 states some open problems which are direct implications of the analytical investigations. Section 6 reports on experiments we have performed to assess the performance of well-known split strategies under the four query models. Some further open problems conclude the paper.

2 The Setting

Let us shortly review the basic concepts of spatial data management. Data structures which efficiently support spatial access to geometric objects cluster these objects in data buckets according to their spatial locations. With each data bucket a subspace of the data space, the so-called bucket region, is associated containing all objects of the corresponding bucket. Except for the BANG-File [2] and the cell tree [3], a bucket region is a multidimensional interval.

Point data structures usually create bucket regions which form a partition of the data space (see e.g. [2, 5, 7]). Data structures for bounding boxes generate bucket regions which may overlap and do not necessarily cover the entire data space (see e.g. [4, 6, 8]).

We are interested in the range query performance of a spatial data structure $DS(G)$, which is currently storing a set G of geometric objects. We restrict our investigations to so-called window queries where the query window forms a d-dimensional interval. The operation *window-query* $(w, DS(G))$ retrieves for the query window w each point object which is located in w, respectively each bounding box which intersects w.

The window query performance of a data structure heavily depends on the kind of the window queries to be performed, and hence depends on the expected user behavior – which is usually unspecified.

A closer look turns out that the user may vary the aspect ratio, the location of the query window and the query value which can be the area of the window or the size (cardinality) of the answer set. Let us assume that there is no direct correlation between these parameters. So we can discuss each parameter in turn.

We assume the aspect ratio to be 1, i.e. square windows. This seems to be appropriate unless some slope bias is known beforehand, since the expected value of the aspect ratio is 1 if all aspect ratios are equally likely.

Concerning the window location, two possible user behaviors seem to be likely:

- Every part of the data space is equally likely to be requested, i.e. a uniform distribution for the window center is assumed. This assumption models the situation where no user preference is known beforehand, respectively reflects the behavior of novice and occasional users.
- Each object is equally likely to be requested. This situation where queries prefer densely populated parts of the data space can be observed in many applications.

Let us now turn to the specification of the query value. In a user-driven query, the requested part of the data space usually has to be represented on a screen. Then two possible variants of user behavior seem to be likely:

- The user specifies the query value in terms of the window area. We assume the area to be a constant which is typical for situations where the user specifies queries such that the requested part covers (more or less) the entire screen (zooming facilities neglected).
- The user always determines the query value with the intention to retrieve the same (constant) number of objects. Here, the cardinality of the answer set is assumed to be constant. This is typical for an experienced user who tends to request an amount of information which is neither overloaded nor insufficient and satisfies his personal needs. In this model, the window area depends on the underlying object population.

Combining the two proposed variants of query value, respectively location, results in four different models for user-defined window queries. In the next section, we render them in precisely defined probabilistic models. Such a framework allows for an analytical characterization of arbitrary data space organizations with respect to the underlying query model.

3 Probabilistic models for user-defined window queries

Let us resume by defining the introduced problem more precisely as follows. Let d be the dimension of the data space we consider, $S_i = [0,1)$, $1 \leq i \leq d$, and $S = S_1 \times S_2 \times \ldots \times S_d$ the d-dimensional data space in which all geometric objects are defined. A geometric object is either a point p given by its coordinates, i.e. $p = (p.x_1, p.x_2, \ldots, p.x_d)$, $p.x_i \in S_i$, or a d-dimensional interval $v = [v.l_1, v.r_1] \times \ldots \times [v.l_d, v.r_d]$, $v.l_i, v.r_i \in S_i$, $v.l_i \leq v.r_i$. An interval v can be interpreted as bounding box of an arbitrary non-point object.

Let us assume that for storing the set G of objects the data structure $DS(G)$ currently consumes m consecutive blocks $B_1, B_2, \ldots, B_m$, the so-called data buckets. Each bucket has a capacity of c objects. With each object g, a bucket is uniquely associated. The bucket

region $R(B_i) \subseteq S$ of a bucket B_i is a d-dimensional interval enclosing (the bounding boxes of) all objects in B_i. For a bucket set $B = \{B_1, \ldots, B_m\}$ we call $R(B) = \{R(B_1), \ldots, R(B_m)\}$ the corresponding organization of the data space.

Let $\mathbb{R}^d$ be the space in which all query windows are defined. The location of a query window w is specified by its center $w.c = (w.l_1 + w.l_2)/2$ componentwise. We call a query window w legal iff $w.c \in S$. Let W be the set of all legal windows. Let $\vec{f_c} : S \rightarrow (\mathbb{R}^+)^d$ be the (componentwise continuous) density function of the center distribution of legal windows and $F_c : S \rightarrow [0,1]$ be the corresponding distribution function. Let $\vec{f_G} : S \rightarrow (\mathbb{R}^+)^d$, resp. $F_G : S \rightarrow [0,1]$, be the (componentwise continuous) density function, resp. (continuous) distribution function, of the location of the geometric objects. We assume that the location of a non-point object g is uniquely determined by a point belonging to g, e.g. the center of g. Let $\mathcal{M} : W \rightarrow [0,1]$ be a probability measure for legal windows.

A window query model $\mathcal{WQM}$ is a 4-tuple with the components aspect ratio ar (which is 1:1 for all models), window measure $\mathcal{M}$, the window value $\mathcal{M}(w)$ which is a constant $c_{\mathcal{M}}$ for all legal windows w, and center distribution F_c:

$$\mathcal{WQM} = (ar, \mathcal{M}, c_{\mathcal{M}}, F_c).$$

Model 1 is characterized by choosing the conventional area function A as window measure $\mathcal{M}$, constant window area c_A for the window value and a uniformly distributed window center:

$$\mathcal{WQM}_1 = (1{:}1, A, c_A, U[S]).$$

In model 2, the window measure is retained but the center distribution equals the object distribution:

$$\mathcal{WQM}_2 = (1{:}1, A, c_A, F_G).$$

In model 3, we assume the window measure $\mathcal{M}$ to be the answer size of the query. Hence, the window measure is $F_W : W \rightarrow [0,1]$ where $F_W(w) = \int_{S \cap w} \vec{f_G}(p)dp$, the window value is some constant c_{F_W}, and the window center is uniformly distributed (Note that the window areas depend on the object distribution F_W.),

$$\mathcal{WQM}_3 = (1{:}1, F_W, c_{F_W}, U[S]).$$

Model 4 is similar to model 3 except for the window center distribution F_c which equals the object distribution F_G:

$$\mathcal{WQM}_4 = (1{:}1, F_W, c_{F_W}, F_G).$$

The four query models presented above provide a framework for the following investigations on window query performance.

4 Analytical results on window query performance

For each of the four window query models, we characterize data space organizations in terms of the expected number of bucket accesses needed to perform a window query. Without loss of generality and only for simplicity reasons, we choose $d = 2$ for further considerations. This reduces bounding boxes, bucket regions, and query windows to two-dimensional rectangles.

For $\mathcal{WQM}_k, 1 \leq k \leq 4$, and data space organization $R(B) = \{R(B_1), \ldots, R(B_m)\}$, let $P_k(w \cap R(B_i) \neq \emptyset)$ be the probability that the window w intersects bucket region $R(B_i)$, and $P_k(w \cap R(B); j)$ be the probability that window w intersects exactly j bucket regions in $R(B)$. Then for a data space organization $R(B)$, the expected number of buckets intersecting a query window in model k – we call it the *performance measure* for model k – is given by

$$\mathcal{PM}(\mathcal{WQM}_k, R(B)) = \sum_{j=0}^{m} j \cdot P_k(w \cap R(B); j).$$

The following Lemma facilitates the computation of $\mathcal{PM}(\mathcal{WQM}_k, R(B))$.

Lemma.

$$\sum_{j=0}^{m} j \cdot P_k(w \cap R(B); j) = \sum_{i=1}^{m} P_k(w \cap R(B_i) \neq \emptyset)$$

Proof (by induction on m). To improve readability we omit index k. We extend the notation $R(B)$ of a data space organization to $R(B^{(m)}), m \in \mathbb{N}_0$, to indicate the cardinality of $R(B)$.

Basis $m = 0$: We get

$$\sum_{j=0}^{0} j \cdot P(w \cap R(B^{(0)}); j) = 0 = \sum_{i=1}^{0} P(w \cap R(B_i) \neq \emptyset).$$

By induction hypothesis

$$\sum_{j=0}^{m} j \cdot P(w \cap R(B^{(m)}); j) = \sum_{i=1}^{m} P(w \cap R(B_i) \neq \emptyset)$$

holds for all $m \in \mathbb{N}$.

Induction step $m \rightarrow m+1$:

We focus on region $R(B_{m+1}) \in R(B^{(m+1)})$. Considering $R(B_{m+1})$ separatly and removing it from $R(B^{(m+1)})$ yields the decreased set $R(B^{(m)}) = R(B^{(m+1)}) \backslash \{R(B_{m+1})\}$. We get

$$\sum_{j=0}^{m+1} j \cdot P(w \cap R(B^{(m+1)}); j)$$

$$
\begin{aligned}
&= \sum_{j=0}^{m+1} j \cdot [P(w \cap R(B^{(m)}); j) \cdot P(w \cap R(B_{m+1}) = \emptyset) \\
&\quad + P(w \cap R(B^{(m)}); j-1) \cdot P(w \cap R(B_{m+1}) \neq \emptyset)] \\
&= \sum_{j=0}^{m+1} j \cdot [(P(w \cap R(B^{(m)}); j-1) - P(w \cap R(B^{(m)}); j)) \\
&\quad \cdot P(w \cap R(B_{m+1}) \neq \emptyset) + P(w \cap R(B^{(m)}); j)] \\
&= P(w \cap R(B_{m+1}) \neq \emptyset) \\
&\quad \cdot \sum_{j=0}^{m+1} j \cdot [P(w \cap R(B^{(m)}); j-1) - P(w \cap R(B^{(m)}); j)] \\
&\quad + \sum_{j=0}^{m+1} j \cdot P(w \cap R(B^{(m)}); j)
\end{aligned}
$$

The second sum meets the induction hypothesis, hence it remains to prove

$$
\sum_{j=0}^{m+1} j \cdot [P(w \cap R(B^{(m)}); j-1) - P(w \cap R(B^{(m)}); j)] = 1.
$$

We have

$$
\begin{aligned}
&\sum_{j=0}^{m+1} j \cdot [P(w \cap R(B^{(m)}); j-1) - P(w \cap R(B^{(m)}); j)] \\
&\quad = \sum_{j=0}^{m} (j+1) \cdot P(w \cap R(B^{(m)}); j) \\
&\quad - \sum_{j=0}^{m+1} j \cdot P(w \cap R(B^{(m)}); j) \\
&\quad = \sum_{j=0}^{m} P(w \cap R(B^{(m)}); j) \\
&\quad = 1
\end{aligned}
$$

□

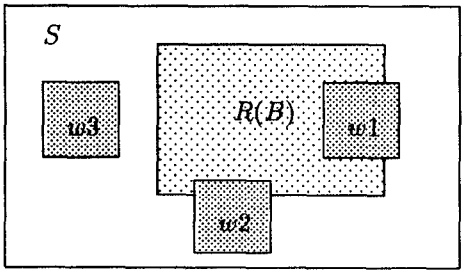

Figure 1: Representatives of query windows according to a bucket region.

The Lemma tells us that for the computation of $\mathcal{PM}(\mathcal{WQM}_k, R(B))$, it remains to compute $P_k(w \cap R(B_i) \neq \emptyset)$ for every bucket region $R(B_i) \in R(B)$. For this purpose, let us consider a single bucket region $R(B_i)$. Every legal window belongs to one of the following three classes:

- Queries with centers inside $R(B_i)$,
- Queries with centers outside $R(B_i)$, but intersecting $R(B_i)$,
- Queries which do not intersect $R(B_i)$.

Figure 1 depicts a representative of each class.

For bucket region $R(B_i)$, let $R_c(B_i)$ be the domain in which the centers of all windows intersecting $R(B_i)$ are located. Hence, the probability that a random window intersects $R(B_i)$ equals the probability that the window center falls into domain $R_c(B_i)$. Obviously, $R_c(B_i)$ depends on the underlying query model. To be precise, $R_c(B_i)$ depends on the window value $c_{\mathcal{M}}$ and so, of course, on the window measure $\mathcal{M}$.

We exemplarily explain the computation of $P_k(w \cap R(B_i) \neq \emptyset)$ for the first model because analogous considerations hold for the remaining models.

In this model, query windows are squares with fixed width and height $\sqrt{c_A}$, and their centers occur at each possible position in the data space with equal probability.

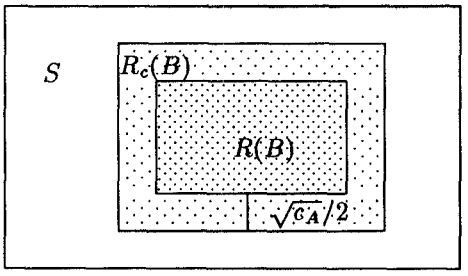

Figure 2: A domain $R_c(B)$ in model 1.

For simplicity, let us assume for a while that each region $R(B_i)$ is far enough off the data space boundaries such that domain $R_c(B_i)$ is just the region $R(B_i)$ inflated by a frame of width $\sqrt{c_A}/2$. If $R(B_i)$ has width $R(B_i).L$ and height $R(B_i).H$ then the probability that w intersects $R(B_i)$ is determined by the area of $R_c(B_i)$ which is $(R(B_i).L + \sqrt{c_A}) \cdot (R(B_i).H + \sqrt{c_A})$. So we get (see figure 2)

$$
\begin{aligned}
&\overline{\mathcal{PM}}(\mathcal{WQM}_1, R(B)) = \\
&\quad = \sum_{i=1}^{m} (R(B_i).L + \sqrt{c_A}) \cdot (R(B_i).H + \sqrt{c_A})
\end{aligned}
$$

$$= \sum_{i=1}^{m} R(B_i).L \cdot R(B_i).H + \sqrt{c_A} \cdot \sum_{i=1}^{m} (R(B_i).L + R(B_i).H) + c_A \cdot m.$$

What do we gain from this simple performance measure? In geometric terms, this function combines the sum of all region areas, the weighted sum of all region perimeters, and the weighted number of regions. It should be mentioned that for the first time the strong influence of the region perimeters is revealed. To our knowledge, so far only in the R*-tree simulations to a certain extent region perimeters have been taken into account [1]. Besides this observation, a number of plausibility arguments can now quantitatively be illustrated. For instance, the term $c_A \cdot m$ tells us that high bucket utilization is a more important factor if query windows are larger. On the other hand, very small query windows make the term $\sum_{i=1}^{m} R(B_i).L \cdot R(B_i).H$ dominate the others. Whenever the data space organization partitions the data space, $\sum_{i=1}^{m} R(B_i).L \cdot R(B_i).H$ equals 1, no matter how regions are chosen. Then for query windows with $c_A \ll R(B_i).L + R(B_i).H$, for any region $R(B_i) \in R(B)$ the term $c_A \cdot m$ is negligible, and the sum of the perimeters determines the efficiency. For query windows ensuring $c_A \gg R(B_i).L + R(B_i).H$, the term $c_A \cdot m$, or in other words the number of buckets, respectively the bucket storage utilization, is the significant part of the formula. Note that the latter arguments substantiate common opinions and experiments in the spatial data field.

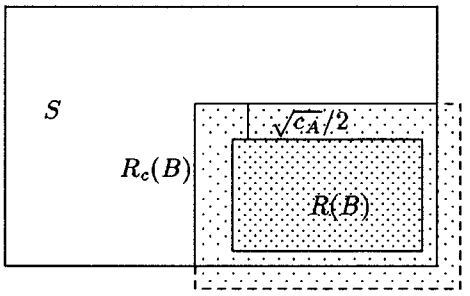

Figure 3: Boundary considerations in model 1.

To get the exact performance measure for model 1 we must take the data space boundaries into account. If a region $R(B_i)$ comes close enough to a data space boundary then domain $R_c(B_i)$ is not just region $R(B_i)$ inflated by a frame of width $\sqrt{c_A}/2$ but the restriction of the inflated $R(B_i)$ to S (see figure 3). (Remember that this holds for each $R_c(B_i)$ by definition.) This leads to

$$\mathcal{PM}(\mathcal{WQM}_1, R(B)) = \sum_{i=1}^{m} A(R_c(B_i)).$$

Analogous considerations lead to the performance measures of the remaining models. In model 2, the domains $R_c(B_i)$ are identical to those in model 1. But instead of simply taking the area of $R_c(B_i)$ – an implication of the uniform window center distribution – $R_c(B_i)$ must now be valued by the window measure F_W:

$$\mathcal{PM}(\mathcal{WQM}_2, R(B)) = \sum_{i=1}^{m} F_W(R_c(B_i)).$$

Along these lines, the performance measure for model 3 can easily be written down as

$$\mathcal{PM}(\mathcal{WQM}_3, R(B)) = \sum_{i=1}^{m} A(R_c(B_i)).$$

However, it is not a trivial task to determine the domains $R_c(B_i)$ in $\mathcal{WQM}_3$. By definition, domain $R_c(B_i)$ is the set of the centers of all legal windows w such that window value $\mathcal{M}(w) = c_{\mathcal{M}}$ and w intersects region $R(B_i)$. Since $c_{\mathcal{M}} = c_{F_W}$, i.e. the answer size is assumed to be constant, the window area depends on the location of the window center, and domain $R_c(B_i)$ has a non-rectilinear shape depending on F_G, although bucket region $R(B_i)$ is always a rectangle. To get an impression of how complicated the determination of the domain $R_c(B_i)$ for a region $R(B_i)$ can be, we provide the following example.

Example. For model 3, we assume a non-uniform but still simple object distribution given by the vector-valued density function $\vec{f_G}(p) = (1, 2p.x_2)$ for $p = (p.x_1, p.x_2)$, and a window value $c_{F_W} = 0.01$. To avoid problems incured by data space boundaries we choose the bucket region $R(B_i) = [0.4, 0.6] \times [0.6, 0.7]$. The area of a query square w depends on the location of its center $w.c$. After some calculations we yield for window w the area $A(w) = \frac{0.01}{2w.c.x_2}$ and the side length $l(w) = \sqrt{A(w)}$. To get the lower boundary of $R_c(B_i)$ we compute the curve of all window centers whose associated window just touch the lower boundary of region $R(B_i)$. Hence, we have to solve for $w.c.x_2$ the equation $0.6 - w.c.x_2 = l(w)/2$. The equations $c.w.x_2 - 0.7 = l(w)/2$, $0.4 - w.c.x_1 = l(w)/2$, $w.c.x_1 - 0.6 = l(w)/2$, respectively, for the upper, left and right boundaries, respectively, are treated analogously. The resulting region $R_c(B_i)$ is depicted in figure 4.

□

The transition from model 3 to model 4 is analogous to the transition from model 1 to model 2. Domains

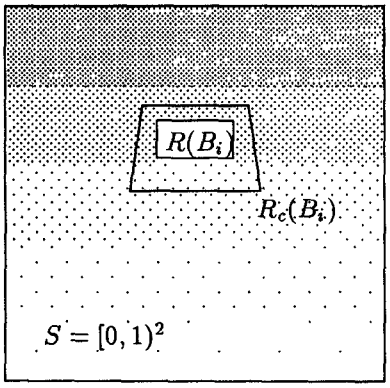

Figure 4: Non-rectilinear domain $R_c(B_i)$ from the example.

$R_c(B_i)$ in model 4 are identical to those in model 3. But as before, instead of taking the areas of the $R_c(B_i)$ as values, in this case the $R_c(B_i)$ must be valued by F_W before summing them up. Altogether we get

$$\mathcal{PM}(\mathcal{WQM}_4, R(B)) = \sum_{i=1}^{m} F_W(R_c(B_i)).$$

5 Obvious questions

In the preceeding section, for each of the four query models we derived a performance measure which characterizes data space organizations in terms of the expected number of bucket accesses needed to perform a random window query. Our investigations give rise to some obvious questions.

For example, for an object set G and a window query model WQM

What is an optimal data space organization? and
Which data structure, resp. corresponding insertion algorithm, achieves an optimal data space organization?

We must admit that we have no answers yet. After all, when addressing the second question we have gained some further insight into spatial data structures we want to report on in the following.

For sake of simplicity, we concentrate on data structures for points. These data structures usually partition the data space, i.e. when an object insertion causes a data bucket overflow the corresponding bucket region $R(B)$ is partitioned by a splitline into two bucket regions $R(B_1)$ and $R(B_2)$ and the objects in $R(B)$ are distributed over the corresponding two buckets. It is important for the efficiency of a spatial data structure that for any data bucket to be split the choice of the split line depends only on the corresponding bucket region, i.e. can be chosen independently of all other bucket regions. This locality criterion for binary splits naturally leads to binary tree directories for the bookkeeping of the binary splits, resp. data space organization.

The following observation is crucial for the data structure performance. As long as the data structure is flexible enough to support arbitrary split strategies, the choice of the split strategy determines the performance efficiency. This perception immediately poses the next question:

For query model k, what is the best binary split strategy?

Unfortunately, we again cannot provide an answer. It is clear, that carrying the optimality criterion of the global situation over to the local situation of a bucket split will not achieve the desired effect. A sound solution will be based on stochastic optimization theory for dynamic processes, and still remains an open problem.

Finally, we end up with the question

How do well-known split strategies perform according to the four query models?

Remember that up to now all performance evaluations have been based only on the first query model. Because of the practical relevance of the models 2, 3, and 4, such an investigation is meaningful and the next section deals with the experiences we gained from our experiments.

6 Experimental results

In order to assess the effects of split strategies used in data structures for points, we have chosen the radix split, the median split and the mean split. As underlying data structure we have taken the LSD-tree [5], whose binary tree directory allows for the realization of arbitrary split strategies, and implemented it in Eiffel on a SUN SPARCstation.

During each test run 50.000 2-dimensional points from the data space $[0,1) \times [0,1)$ have been inserted into the initially empty LSD-tree. In order to achieve statistically significant results (a small confidence interval) the bucket capacity was set to $c = 500$ objects. Whenever a split has to be performed, the split line is chosen such that it hits the longer bucket side and the hit position is defined by the underlying split strategy. The three split strategies were evaluated for each query model assuming two constants $c_{\mathcal{M}} = 0.01$ and $c_{\mathcal{M}} = 0.0001$. For models 3 and 4, the performance measures are computed by an approximation procedure. For each bucket split, the number of objects currently being stored and the according performance measures are reported.

A β-distribution randomly generates different object distributions, namely a uniform, a 1-heap and a 2-heap distribution. The relatively extreme population of the 1-heap distribution usually exhibits certain effects very clearly, while the 2-heap distribution is a suitable abstraction of cluster patterns typically occuring in real applications. A representative pattern of each of

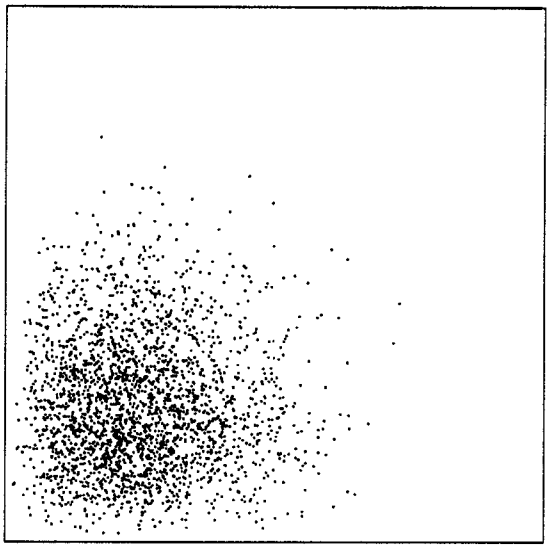

Figure 5: 1-heap distribution.

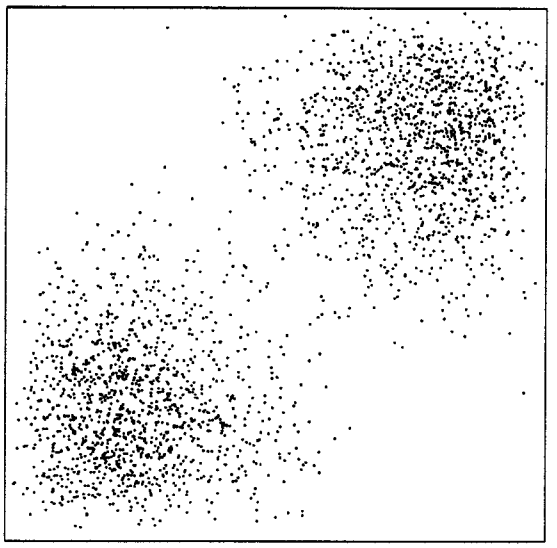

Figure 6: 2-heap distribution.

the heap distributions is depicted in figures 5 and 6, respectively.

Let us start the discussion of the experimental results with the main outcome of our extensive simulations. The efficiencies of the data space organizations created by the three split strategies differ only marginally. Differences mainly depend on the point of time the snapshot was performed and never exceed more than ten percent of the absolute values. Hence, we restrict our further discussion to radix splits.

Figure 7, respecively 8, depicts the different performance measures for the 1-heap, resp. 2-heap, distribution with respect to each model and for $c_{\mathcal{M}} = 0.01$. It turns out that the different model assumptions lead to rather different evaluations of the same data space partition. This effect is mainly observed for distributions with a zero population in wide parts of the data space like e.g. the 1-heap distribution. Note, however, that for a direct comparison the absolute values must be related to the answer size.

A second bunch of simulations deals with the situation where the insertion sequence of objects is somewhat "presorted". Such a presorting often occurs in real applications. For example, whenever we have used real geographic data in what application so ever, the data file was "sorted" according to counties, municipalities or districts, while each data pile itself was almost random. In order to cover this situation by our experiments, we take the 2-heap distribution and completely insert the one heap first and then the other heap, both in random order.

Again, our experiments do not exhibit significant differences for the different split strategies. This result is somewhat unexpected because the radix split is well known for its robustness while especially the median split is known to be order sensitive. Even in the situation when the first heap has been inserted and the procedure switches to the second heap, for none of the three split strategies a significant deterioration can be observed. For an entire discussion, however, it should be mentioned, that in case of the median split the directory tends to a certain degeneration.

Although all split strategies create data space organizations of more or less the same efficiency, our personal choice is the radix split. Besides the robustness of the directory against insertion ordering, the entries in the directory, i.e. the split position, can be encoded with short bitstrings thus keeping the directory small.

Another outcome of our experiments not mentioned so far is the effect of using minimal bucket regions. These regions are not bounded by split lines or data space boundaries but are just the bounding boxes of the objects actually stored in the corresponding buckets. It turns out that for small window values $c_{\mathcal{M}}$, minimal bucket regions can improve the performance up to 50 percent.

7 Open problems

The following open problems can be added to the list presented in section 5.

It seems to be natural to extend the search for efficient split strategies to data structures for non-

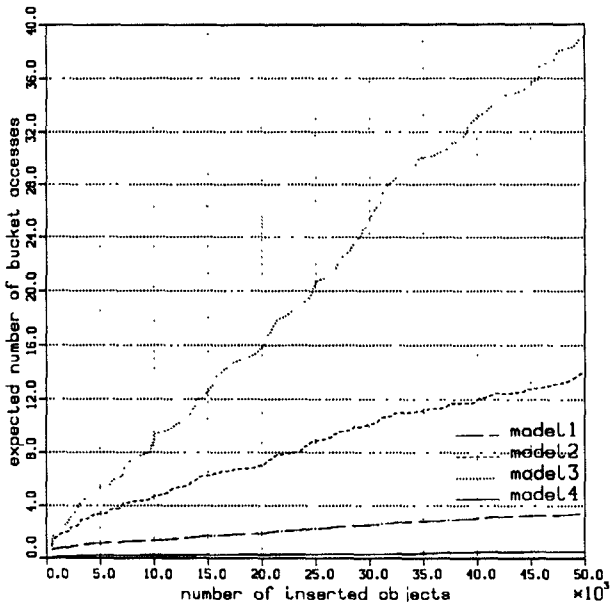

Figure 7: The four performance measures for 1-heap distribution, radix splits and $c_{\mathcal{M}} = 0.01$.

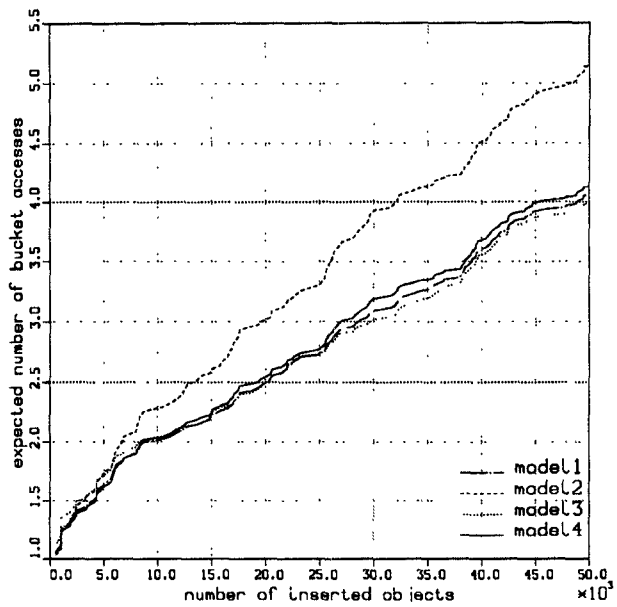

Figure 8: The four performance measures for 2-heap distribution, radix splits and $c_{\mathcal{M}} = 0.01$.

point geometric objects. These data structures generate bucket regions which may overlap and do not necessarily cover the entire data space. For example, it should be worthwile to use the knowledge gained from our analytical investigations for an improvement of the split strategies of the R-tree which are not well understood yet.

Although the time penalty incured by external directory accesses is small compared to data bucket accesses, it would be desirable (at least from a theoretical viewpoint) to extend the performance measures to cover external directory accesses as well. Usually, with each directory page a directory page region is associated which is the bounding box of all data bucket regions pointed at from the directory page (see e.g. [4, 5]). Since directory page regions again form a data space organization, such an integrated analysis of range query performance seems to be feasible.

Finally, the development of analogous performance measures for other query types, like e.g. nearest neighbor queries or queries partly focussing on the volume (area) of the objects, would improve the understanding of spatial data structures even more.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Atlantic City, 1990.

[2] M.W. Freeston. The BANG file: a new kind of grid file. In *Proc. ACM SIGMOD Int. Conf. on the Management of Data*, pages 260–169, San Francisco, 1987.

[3] O. Günther. *Efficient structures for geometric data management*, volume 337 of *Lecture Notes in Computer Science*. Springer, Berlin, 1988.

[4] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, Boston, 1984.

[5] A. Henrich, H.-W. Six, and P. Widmayer. The LSD-tree: spatial access to multidimensional point- and non-point objects. In *15th Int. Conf. on VLDB*, pages 45–53, Amsterdam, 1989.

[6] A. Hutflesz, H.-W. Six, and P. Widmayer. The R-file: an efficient access structure for proximity queries. In *Proc. 6th Int. Conf. on Data Engineering*, pages 372–379, Los Angeles, 1990.

[7] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984.

[8] B. Seeger and H.-P. Kriegel. The buddy-tree: an efficient and robust access method for spatial data base systems. In *16th Int. Conf. on VLDB*, pages 590–601, Brisbane, 1990.