A class of R-tree histograms for spatial databases

Technical Report

Daniar Achakeev
Department of Mathematics and Computer Science
Philipps-Universität Marburg, Germany
achakeye@mathematik.uni-marburg.de

Bernhard Seeger
Department of Mathematics and Computer Science
Philipps-Universität Marburg, Germany
seeger@mathematik.uni-marburg.de

ABSTRACT

Spatial histograms are extremely useful for approximate query processing in large spatial databases. The problem of generating optimal spatial histograms is NP-hard; therefore, many heuristic-based methods have emerged over the last 15 years. Shortcomings of these methods are their complex algorithmic design and their sensitivity to parameter setting, preventing easy integration into real systems.

In this paper, we present a class of spatial histograms derived from the popular family of R-tree indexes. We propose a cost-optimized approach that combines bulk-loading of R-trees and construction of spatial histograms. This creates a robust histogram method with high accuracy for selectivity estimation of spatial queries. In particular, the estimation error continuously decreases with increasing number of histogram buckets, and therefore, our histogram methods benefit from a large number of histogram buckets.

For experimental evaluation, we compare the performance of our methods with state-of-the-art spatial histograms. In contrast to previously conducted experiments, we examine the performance under different classes of workloads. Our results confirm that our histograms display low estimation errors and can be built fast. In addition, their performance is very robust under different workloads.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms

Algorithms, Performance

Keywords

Histogram, R-tree, Spatial Query Selectivity Estimation

1. INTRODUCTION

Histograms are important data structures primarily used in database systems for estimating the selectivity of queries. They are also applied to obtaining quick approximate response for aggregate queries. While one-dimensional histograms are widely available in almost all database system, only a very few systems offer multidimensional histograms. Most of them are simple grid-based methods that are applicable to two-dimensional point data only. These methods perform poorly on rectangle data or when the independence assumption of the attributes is violated.

The design of efficient multidimensional histograms turns out to be much harder already for the two-dimensional case. In fact, the problem of designing optimal multidimensional histograms is known to be NP-hard. Therefore, many heuristics have been developed and evaluated in various experimental settings. In general, these heuristics result in fairly complex parameterized algorithms with a runtime often substantially higher than the runtime of the one-dimensional counterparts. This is often not acceptable because histograms have to be rebuilt quite frequently. In addition, the algorithms are often quite sensitive to small variances of the parameter values.

In this paper, we revisit the problem of designing efficient multidimensional histograms from the perspective of bulk-loading spatial index-structures, e.g., R-trees. Similar to R-trees, a histogram is viewed as a set of bounding boxes, but each of them is associated with statistical indicators like the number of spatial objects that are assigned to the box. Rather than directly generating histogram buckets, our method relies on a two-step approach: First, the leaf level of an R-tree is generated and second, adjacent leaves are merged into larger histogram buckets. Crucial and sensitive parameters are avoided; instead both steps rely on the optimization of a widely accepted cost function. This makes our approach very appealing to an end-user.

Even though our optimization bases on minimizing a cost function, it still remains a heuristics like it is for all other multidimensional histograms. It is therefore of utmost importance to use a thoroughly designed experimental setup to provide a meaningful and fair comparison with competitors. So far, there is no commonly agreed experimental setup for spatial and multidimensional histograms. In particular, we found serve deficiencies in current experimental work, e.g., small data sets, low selectivity of queries, uniformly distributed queries.

Our contributions are summarized as follows:

1. We present a uniform rectangle partitioning framework for R-tree loading and histogram construction. Derived from this framework, we present an efficient two-step approach to generating multidimensional histograms.
2. We introduce query models for workload generation and examine the accuracy of histograms under these workloads.
3. We present an experimental performance comparison of a large number of multidimensional histograms.

The paper is organized as follows: In Section 2, we discuss related work. Preliminaries like our underlying cost model

are introduced in Section 3. In Section 4, we present our uniform framework for R-tree loading and histogram construction. In Section 5, we introduce the query models and report the most important results of an experimental evaluation of our methods in comparison to related histogram methods. The paper concludes with a summary of the most important results.

2. RELATED WORK

During the last three decades one-dimensional histograms have been used widely for the purpose of selectivity estimation and with a fair amount of success [15, 9]. Nevertheless, in case of multidimensional histograms, we are still facing many challenges, that need to be solved[8]. To tackle these problems, many different heuristic based methods were proposed. All of them aim to partition the multidimensional data in rectangular buckets for a given space budget. The data within buckets is uniformly distributed, since the query estimation relies on uniform data distribution assumption. The heuristic methods are motivated by the results in [12]. The authors show that computing the non-overlapping rectangular partitioning with near-uniform data distribution within buckets is NP-hard.

One of the first methods proposed for multidimensional data is *hTree* [11]. It constructs non-overlapping partitioning of multidimensional space based on a frequency as source parameter. Only one dimension is approached at a time and partitioned in buckets with an identical number of objects, resulting in a *equi-depth* histogram. The advantage of hTree is its low construction cost. However, the partitioning rule is too rigid for highly skewed data. In contrast, *mHist* uses space partitioning. Space is partitioned along the dimension that benefits most from a split. The split decision is made based on a marginal frequency distribution. This approach was developed for relational data and focuses mainly on approximating point frequencies. However, selectivity estimation for spatial data differs from traditional one [1]. The object frequencies may be uniform, but the locations can be highly skewed, and the objects vary in sizes and shapes.

To provide accurate estimation for spatial objects and also I/O efficiency, the *MinSkew*-Histogram method was proposed [1]. The authors proposed two construction strategies. The basic variant works as follows: in the first phase, the algorithm computes a regular grid and stores the number of intersecting spatial objects for each cell. Based on the computed grid, the recursive binary space partitioning (BSP) is used for histogram computation. The buckets are picked for further processing based on a split value that will lead to greatest reduction of data skew. The decision is local, so that for all dimensions, all possible cuts based on marginal objects frequencies are considered. The authors observed that a fixed grid size is sensitive to the size of queries [1] (high grid resolution favors small sized queries and small resolution large queries). To lessen this effect the second construction strategy *MinSkew-Progressive-Refinement* utilizes grids with different resolutions. Each grid resolution is used to construct the equal portion of histogram buckets. The computation is processed in top-down fashion starting with a low resolution grid applying BSP in each step. The downside of both strategies is that the performance is sensitive to the grid resolutions.

GenHist proposed by [7] tries to identify high density regions. In contrast to the previous methods, the bucket rectangles may overlap. Moreover, the buckets can be contained in other buckets. GenHist finds regions with high object density, excises them but leaves enough data in the parent bucket so that the parent buckets distribution flattens. Again, the method uses a regular grid as a starting point for histogram construction.

The recently proposed method *STHist* [16] applies the idea of *GenHist* to 2-3-dimensional spatial objects. In the basic variant decision about whether the region is dense is made by applying a sliding window over all dimensions, approximating the frequency distribution by a marginal distribution. The dense regions called *Hot-Spots* build hierarchies, so that the Histogram is represented as an unbalanced R-tree. In the advanced variant called *STForest*, the algorithm first computes coarse partitions according to the object skew, and then applies a sliding window algorithm to them. The idea behind this is that if the region is already uniformly distributed further partitioning is unnecessary. Moreover, the coarse regions merge together if the skew of merged bucket decrease. The experiments conducted in [16] show that STHist is superior to other proposed methods. However, STHist has time complexity $O(n^2)$ for 2-dimensional and $O(n^3)$ for 3-dimensional data.

Recently, the class of self-tuning histograms like *STHoles* and *ISOMER* were proposed [4, 18]. In general these methods incrementally update buckets and their frequency information, using query feedback. These kind of methods is very appealing, because the incremental modification the histogram adapts to the real distribution of a data. Moreover, the methods can be applied independently on top of different approaches.

Another way to obtain a spatial histogram is to generate it using a spatial index structure like R-tree [1, 8, 5]. The recently proposed approach *rKHist* [5] uses this idea and is based on R-tree bulk-loading procedure[10]. The data is presorted according the Hilbert space-filling-curve. After the leaf nodes are generated, one possibility to generate a histogram is to pack nodes according to the sorting order in equi-sized histogram buckets. This leads not always to a good partitioning. Especially, for near-uniformly and uniformly distributed data equi-sized partitioning wastes buckets for regions with a high object density and yield high overlap, despite the fact that the regions have uniform distribution[1]. Therefore, the authors proposed a greedy algorithm that utilizes a sliding window of pages along the Hilbert order. The algorithm is parametrized with a number of buckets that should be considered for a splitting. A bucket-split is applied if it leads to an improvement according to the proposed cost function.

Our approach differs from rKHist in that we tune the R-trees according to the widely used R-tree cost model. Our generic sort-partition framework computes optimal partitioning for a given cost function according to the sorting order of rectangles. The framework relies on the dynamic programming scheme proposed by [9] for generating one dimensional V-optimal histograms.

The scheme proposed in [9] is also used in [20] for computing a set of k minimal bounding rectangles (MBR) from a 2-dimensional point set. The goal was to reduce communication costs for mobile devices by approximating the spatial query result by a set of MBRs with a minimal information loss f_i. The authors showed that computing such representations is NP-hard even for d=2. One of their heuristics

first sorts the query output using the Hilbert order and then apply the partitioning method of [9]. Multi-dimensional histograms and representation with a minimal information loss f_i are related, since both techniques are considered as data summarization methods. In contrast to histograms the optimization function is different and space constraints are disregarded.

In this work, we adapt the dynamic programming scheme [9] for a R-tree based histogram generation introducing the space constraints on bucket capacity size. This allows to generalize the partitioning scheme and to design new more efficient algorithms for R-tree and R-tree based histogram generation. We show that especially for highly skewed data, the R-tree methods return more accurate results. Moreover, R-tree histograms constructed using our dynamic programming framework display good estimation accuracy for near-uniform and uniform data sets.

3. PRELIMINARIES

In this paper, we investigate the problem of histograms construction based on R-trees for a d-dimensional set of N rectangles $\{r_1, \ldots, r_N\}$. The basis for our development of R-tree histograms is sort-based algorithm for bulk-loading of R-tree. Further, we give a brief overview about bulk-loading and a cost model for R-tree, since our histogram relies on R-tree tuned according to it. We assume that R-trees consist of pages with maximum capacity B and minimum occupation $b \leq \lceil B/2 \rceil$. Our description will address the case $d = 2$; the generalization for $d > 2$ is only discussed when necessary.

3.1 Sort-Based Bulk-Loading

Roussopoulos and Leifker [17] introduced the problem of R-tree loading from scratch and presented a sort-based loading technique with complexity $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$, where M denotes the available main memory. After sorting the rectangles according to a one-dimensional criterion, an R-tree can be built bottom-up like it is known from B+-trees. Because the sorting order has a considerable impact on the search efficiency, Kamel and Faloutsos [10] proposed a double transformation: first a rectangle is mapped to a multidimensional point. Then a space-filling curve (SFC) like the Hilbert- or Z-curve is used to generate a one-dimensional value. Hilbert order yield slightly better results, however, Z-curve is simpler to compute and process [13].

The query performance can be estimated using the cost model [10, 14, 19]. We briefly discuss this model in the following. The range queries can be classified by aspect ratio, location (either data distributed or uniform) and size (either volume or number of results) [19]. Assuming an aspect ratio of 1:1 yields four different query models. The simplest query model refers to the uniform distribution of query rectangles with an equal area.

Assume that the domain corresponds to the two-dimensional unit square $[0, 1)^2$. A rectangle $r_i = (cx_i, cy_i, dx_i, dy_i)$ is represented by its center (cx_i, cy_i) and its extension (dx_i, dy_i). For a window query $WQ_{q,s}$ given by its center $q = (qx, qy)$ and its extension $s = (sx, sy)$, the probability that a rectangle r_i intersects the window is $(dx_i + sx) \cdot (dy_i + sy)$. The average number of rectangles intersecting the query window is then given by $\sum_{i=1}^{N} (dx_i + sx) \cdot (dy_i + sy)$. Note that for point queries with $s = (0, 0)$, the equation computes the sum of MBR volumes. By applying this function to the set of leaf bounding boxes, we obtain the expected number of leaf accesses. This is a typical performance indicator for R-trees.

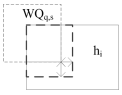

Figure 1: Range query estimation.

3.2 Histogram

We define the output histogram H as a set of buckets $h_1, \ldots, h_m$. The bucket h_i contains statistical information about the set of spatial objects (rectangles) $R_i = r_1, \ldots, r_n$. These are [1]: $MBR(h_i)$ of set R_i, number of elements n_i, average rectangle side length dx^i_{avg}, dy^i_{avg} over set R_i and spatial density information $s_i = Area(h_i)/Area(MBR_{h_i})$. Where, $Area(h_i)$ is defined a sum of rectangle areas in R_i and $Area(MBR_{h_i})$ is defined as an area of bucket MBR.

In this paper, we use notion bucket and MBR as synonyms depending on the context. Further, our goal is to build the histogram in such way that the number of elements referenced in each bucket varies only by a small constant factor, so that H is close to an equi-depth histogram.

The selectivity estimation $est(q)$ for range and point queries is computed based on the uniform distribution assumption. The selectivity estimation of a point query is computed as follows: Let MBR_{h_i} be the bucket MBR containing a query point. Then s_i is an average number of rectangles hit by given point query in the bucket h_i.

Consider a range query $WQ_{q,s}$. Let MBR_{h_i} be a bucket MBR overlapping with the query $WQ_{q,s}$. Let $r_s = MBR_{h_i} \cap WQ_{q,s}$ be an intersection rectangle. r_s is represented by its center (cx_{r_s}, cy_{r_s}) and its extensions (dx_{r_s}, dy_{r_s}). We extend then (dx_{r_s}, dy_{r_s}) with $2dx^i_{avg}, 2dy^i_{avg}$ in both dimensions with a constraint that the extended sides cannot cross the boundaries of MBR_{h_i} (see Figure 1). Then $n_i \cdot \frac{Area(r_s)}{Area(MBR_{h_i})}$ is the estimated number of rectangles intersecting $WQ_{q,s}$ [1].

4. R-TREE FRAMEWORK

In order to obtain a high-quality histogram H, the data should be partitioned in such way that the data within each histogram bucket is near-uniformly distributed. Computing such partitionings is a non-trivial task and in general NP-hard [12]. Furthermore, the way how data is partitioned also influences the quality of the R-tree. Partitionings minimizing sum of MBR volumes yield better R-trees according to the cost model [19]. In order to obtain a partitioning in polynomial time, we use a heuristic method based on SFC. Our approach can be summarized as follows: reduce the complexity of multidimensional partitioning by sorting the data according to a SFC, and solve the partitioning problem optimally for the sorted set. In the following, we present a high level description of the building blocks of our framework:

Sort-Partitioning: Sort the rectangles with respect to a SFC. Partition the sorted sequence optimally according to a cost function C into subsequences of size between b and B.

Bulk-Loading R-tree: *Step 1. Node Generation:* Run *Sort-Partitioning* with parameter settings for b and B according to a given page size. *Step 2. Generation of Index Entries:* For each page, compute the bounding box of its partitions and create the corresponding index entry. *Step 3. Recursion:* If the total number of index entries is less than

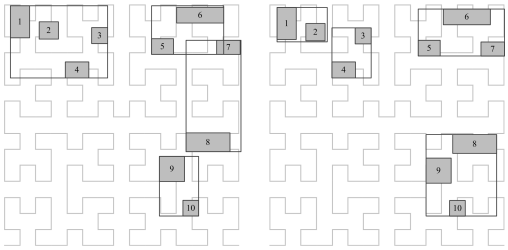

Figure 2: Example of storage bounded partitioning $b = 2, B = 4$ and $m = 4$.

B, store them in a newly allocated root. Otherwise, start the algorithm with the index entries (bounding boxes) from Step 2.

Construction of Histogram H: Run step *Node Generation* of bulk-loading. Collect and store statics. Run *Sort-Partitioning* on generated leaf nodes.

Sort-Partitioning is the crucial step in the algorithm for bulk-loading indexes and generating histograms. The first step uses SFC to sort the data. The second step partitions the sorted sequence of rectangles in optimal way according to a cost function C. Note that our approach is a heuristic and rely on the specific sorting order. The difference between loading indexes and generating histograms is that histograms do not require a recursive processing.

4.1 Details of Sort-Partitioning

In the following, we review the problem of optimal partitioning of a sorted set of rectangles $r_1, \ldots, r_N$ according to a cost function C. Every bucket corresponds to a contiguous subsequence $p_{i,j} = r_i, \ldots, r_j$ such that $b \leq j - i + 1 \leq B$ is satisfied. A valid partition P consists of the subsequences $p_{i,j}$ such that each rectangle belongs to exactly one of them. Let S_N denote the set of all valid partitions and let $S_{N,m}$ be the partitions that consist of exactly m buckets, where $N/b \leq m \leq N/B$ is satisfied. While the standard sort-based bulk-loading strategy stores $\Theta(B)$ rectangles per page (to guarantee that the minimum fan-out is $\Theta(B)$), we do not require such a strict lower bound in the histogram buckets. This increases the flexibility to optimize the partition according to a given cost function. Let $MBR(p)$ be the bounding box of a contiguous sequence p of rectangles. The basic cost function is equal to the sum of volumes of MBR. We refer to this function as C_V.

Figure 2 depicts two possible partitioning with different C_V. Data rectangles are processed according to Hilbert-Curve with $b = 2, B = 4$ and $m = 4$. The center point of rectangle is used for mapping to the Hilbert key. The partitioning on the right has lower C_V costs than the partitioning on the left.

This cost function can be extended to another cost function C_{QP} according to the R-tree cost model, if the average size of queries is known in advance.
$C_{QP} = \sum_{p \in S} area^+(MBR(p), QP)$ be the sum of areas extended by the average side length of the queries. More formally, $area^+(r, QP) = (dx + sx) \cdot (dy + sy)$ for a rectangle $r = (cx, cy, dx, dy)$.

While we have different options for cost functions, we use C(S) to denote to one of them. We consider the following optimization problems:

1. **Storage-bounded partitioning:** Compute a partition $Sm_{opt} \in S_{N,m}$ that minimizes the cost function for the set $\{MBR(p) | p \in S, S \in S_{N,m}\}$.
2. **Bounded partitioning:** Compute a partition $S_{opt} \in S_N$ that minimizes the cost function for the set $\{MBR(p) | p \in S, S \in S_N\}$.

Storage-bounded partitioning allows to choose the desired storage utilization in advance by setting m, while the number of buckets for a bounded-partitioning only satisfies $N/B \leq m \leq N/b$. The first method is very appealing for histograms as generally a space budget is given. Bounding partitioning is interesting for generating optimal R-trees for a cost function C that consist of a sum of positive summands, each corresponds to the partial cost of a rectangle. This allows for the design of efficient algorithms to compute the optimum for both partitioning problems. The basic idea is to use the paradigm of dynamic programming in a similar way as it has been applied to computing optimal one-dimensional histograms [9].

For partitioning the first i rectangles into k contiguous sequences, the computation of the minimum cost $opt^*(i, k)$ can be expressed by the following recursion:

$$opt^*(i, k) = \min_{b \leq j \leq B} \{opt^*(i - j, k - 1) + C(p_{i-j+1,i})\} \quad (1)$$

In general, the $opt^*(i, k)$ function corresponds to the optimal one-dimensional histograms computation proposed by [9] if we use set $b = 1$ and $B = N - m - 1$. In order to compute $opt^*(N, m)$, we apply the recursive formula for all $1 \leq i \leq N$ and $1 \leq k \leq m$, in increasing order of k, and for any fixed k, in increasing order of i. We store all computed values of the $opt^*(i, k)$ in a table (see Alg. 2). Thus, when a new $opt^*(i', k')$ is calculated using Equation 1, any $opt^*(i, k)$ that may be needed can be read from the table. After computation of the optimal cost, we can read the contiguous sequences of the input rectangles out from the dynamic programming table. From this procedure, we obtain the following result.

THEOREM 1. *The optimal partition $S_{N,m}$ of N rectangles into m buckets, each of them containing between b and B contiguous rectangles, can be computed in $O(N \cdot m \cdot B)$ time and $O(N \cdot m)$ space.*

Next, let us consider the problem of bounded partitioning without user-defined storage utilization. At first glance, the problem appears to be harder because the solution space is larger. However, the opposite is true because the parameter m has no effect on the optimal solution anymore. This results in the following simplified recursion:

$$gopt^*(i) = \min_{b \leq j \leq B} \{gopt^*(i - j) + C(p_{i-j+1,i})\} \quad (2)$$

In order to compute $gopt^*(N)$, we compute the recursive formula for all $1 \leq i \leq N$ in increasing order of i. We store all computed values of the $gopt^*(i)$ in a table (see Alg. 2). Thus, when a new $gopt^*(i)$ is calculated using Equation 2, any $opt^*(i)$ that may be needed can be read from the table. As in case opt^* we obtain the result sequence from the table. Thus,

THEOREM 2. *The optimal partition S_N of N rectangles into buckets, each of them containing between b and B contiguous rectangles, can be computed in $O(N \cdot B)$ time and $O(N)$ space.*

Algorithm 1: $opt^*(i,k)$

Input: N rectangles, C cost function, m, b, B
Output: $cost[1 \dots N][1 \dots m]$ cost array

allocate cost array, and initialize for one node;
$cost[\,][\,]$;
for $i = b$ *to* B **do**
  $cost[i][1] = C(1, i)$;
compute best costs for m nodes starting from 2 ;
for $y = 2$ *to* m **do**
  assignment to y pages ;
  for $x = y \cdot b$ *to* $\min(y \cdot B, N)$ **do**
    $s[b \dots B] = 0$;
    max number of entries per node;
    $max_B = (x - B > 0)?B : x - B + 1$;
    for $l = b$ *to* max_B **do**
      $s[l] = cost[x - l][y - 1] + C(x - l, l)$;
    $cost[x][y] = \min(s)$;
return $cost[\,][\,]$;

The result of Theorem 2 shows that loading the optimal leaf level of an R-tree is possible in as little as linear time. The required CPU-time of the method is much lower compared to the optimal solution of storage-bounded loading.

In the following, we give some details that should be taken into account in case of processing a large set of rectangles. Because of the quadratic space required for computing opt^*, it is unlikely that the whole intermediate data sets can be processed in memory. In this case, the data set is processed as follows: we cut the data in sufficient big equi-sized chunks, which can be processed in memory and apply opt^* on each of them independently. In our experiments, we observed that B^2 (where B is equal the number of rectangles in a page) is already sufficient to produce a nearly optimal histogram. For the computation of $gopt^*$, the same strategy can be applied. However, since only the last B entires are required by $gopt^*$, a buffer of B entries is sufficient for processing. If necessary, the partitioning information can be written to external storage. Hence, the overall I/O time is dominated by external sorting.

Algorithm 2: $gopt^*(i)$

Input: N rectangles, C cost function, b, B
Output: $cost[1 \dots N]$ cost array

$cost[\,]$ *allocate cost array, precompute costs for 1 to B elements* ;
for $t = 2b$ *to* N **do**
  $cost[t] \leftarrow \infty$, $R[\,] \leftarrow$ *precompute MBRs for t-B to t-b* ;
  for $l = B$ *to* b **do**
    compute cost for last b to B elements if t-l > b;
    $c_p \leftarrow$ *compute MBR costs* $C(R[l - b])$;
    $c_p \leftarrow cost[t - l] + c_p$;
    if $c_p < cost[t]$ **then**
      $cost[t] = c_p$;
return $cost[\,]$;

4.2 R-tree Histogram

The histogram H_v of our framework is constructed as follows:

Micro-Clustering Step:(we use the same terminology as in [2]) First, the leaf pages of an R-tree are generated using bounded partitioning $gopt^*$. The following parameter are applied: space-filling curve, bucket capacity parameters b and B and cost function C. As a default, we use the Hilbert ordering and the default cost function C_V (minimizing the volume of the bounding boxes). The parameters B and b of the initial step are adjusted to the system physical page size. Additionally, we compute values needed for estimation such as number of elements per leaf, average extents length dx, dy of rectangles inside the leaf (in case of rectangles data) and a sum of their volumes.

Histogram-Generation Step: This step has only one parameter number of buckets m. We apply storage-bounded partitioning opt^* to generate the final m buckets of histogram H_v on the leaf pages (buckets) produced from the first step. The minimal and maximal bucket capacity b and B are depending on parameter m. Therefore, only parameter m has to be set by the user. We compute values $n_i, dx^i_{avg}, dy^i_{avg}, s_i$ based on the information provided from the first step (in case of point data s_i is obtained based on leafs MBR). If there is not enough memory to apply storage-bounded partitioning, we first generate chunks of equal size and apply opt^* to every chunk.

We process the Micro-Clustering step for the following reasons: first, it reduces the time complexity of the final histogram construction. Second, the histogram is generated simultaneously with R-tree index. Third, it can be implemented within same bulk-loading routine.

Since m is an only user parameter for the Histogram-Generation step, the processing time depends also on bucket capacity values b, B for opt^* algorithm. In particular, the inner loop execution time of opt^* (see Alg. 1) algorithm is defined by $B - b$ value. Let N_1 be a number of leaf pages generated by the Micro-Clustering step. Let us consider two extreme cases: fixed sized partitioning $b = B = N_1/m$ and setting $b = 1, B = N_1 - m - 1$ (this corresponds to the original method proposed in [9]). The first setting can be processed in linear time with no quality guarantee. The second needs $O(N_1^2 \cdot m)$ steps to find a minimal cost partitioning.

Since there is the general trade-off between quality and time complexity, we set the bucket capacity parameter b and B for the Histogram-Generation Step as follows: N_1/m is the average number of pages referenced by a bucket for m-bucket histogram. Then we set $b = \max(\lfloor N_1/2m \rfloor, 1)$, $B = \lceil N_1/m \rceil + b$. Thus, the inner loop has time complexity $O(N_1/m)$ and allows some degree of freedom to find a better partitioning according to the cost function.

5. EXPERIMENTS

In this section, we present summarized results obtained from a set of experiments under different query workloads. First, we describe the underlying query models, and then we provide details about our data sets and query files. Finally, we present a detailed discussion of the results.

5.1 Query Models

For experimental settings we followed a methodology for generating workloads based on query models originally proposed in [14] for the design of multidimensional index struc-

tures. The authors classified range queries according to the indicators *aspect ratio*, *location* and *size*. The query size is defined by either area (relative to the entire data space) or the number of qualified objects. Query location can follow either a uniform distribution or the distribution of the underlying data. The aspect ratio equals the width-to-height ratio of the query rectangle, which we assume to be 1 (quadratic windows) in the following. This yields in four different query models:

- M_1: size = area, location = uniform distribution,
- M_2: size = area, location = data distribution,
- M_3: size = number of answers, location = uniform distribution,
- M_4: size = number of answers, location = data distribution.

5.1.1 Data and Query Sets

In our experiments, we adapted the test framework developed for RR*-tree evaluation [3]. The framework consists of 28 different data sets, either points or rectangles, that belong to eight groups *abs, bit, dia, par, ped, pha, uni, rea.* Each of the first seven groups contain three artificially generated data sets with 2,3, and 9-dimensional data following the same distribution in each dimension. Each of the artificial data sets contains at least 1 million objects from $[0,1]^d$. For example, the group *uni* consists of 3 files of 1'000'000 two-, three- and nine-dimensional uniformly distributed points. We give a brief overview about the data sets; 2-dimensional data sets can be roughly grouped in two groups point sets and rectangular sets. Data set *abs* consists of equal sized squares generated from equidistant distribution. Data set *bit* is a point distribution generated according the power low and closely related to Zipf-distribution. Data set *dia* consists of rectangles distributed along the main diagonal. Data set *par* represents a rectangular distribution with a high variance of the size and the shape of rectangles. *ped* is a point distribution of a thin stripped clusters obtained from a data set *par*. Data set *pha* is a set of a ellipse shaped clusters of points generated from data set *par*. Data set *uni* is a uniform point distribution. The eighth group *rea* contains seven real data sets with 2,3,5,9,16,22, and 26 dimensions, respectively. For example, the 2-dimensional data set consists of 1'888'012 bounding boxes of streets of California. The 3-dimensional data set is contains 11'958'999 points from a biological application. The data sources as well as a full description of the data sets are available from [3].

We present in this paper only the results of the 2- and 3-dimensional data sets. According to query models $M_1, \ldots, M_4$, we generated two workloads for each data set and each query model. Two query sets are generated from model M_1. The first one consists of 10'000 uniformly distributed quadratic query rectangles with average volume $V = 0.01\%$ (so that under uniform distribution approximatly 100 objects qualify for a data set with $1'000'000$ objects). The side length of the rectangles are uniformly distributed in range $\left[\frac{1}{2}V^{1/d}, \frac{3}{2}V^{1/d}\right]$. The second query set is generated in the same way with an average volume of 0.1% and consists of 3'164 query rectangles.

The location of the queries from model M_2 follow the underlying data distribution. Again the average volume of the query sets were set to 0.01% and 0.1%, respectively. For the production of queries of model M_3 we first generated uniformaly distributed points and used them for issuing k-nearest neighbor queries with the maximum norm L_∞. The bounding boxes of these k-NN queries, k = 100 and k = 1'000, are used for two sets of window queries with 100 and 1'000 answers per query, respectively. For model M_4, we used the underlying data distribution for producing the reference points for the nearest neighbor queries. Thus, the location of the window queries also follows the data distribution. Again two query sets are generated with 100 and 1'000 answers per query.

Cost Func.	Description
C_V	volume of MBR
C_{QP}	C_V extended by avg. query side lengths
C_{RK}	k-Uniformity metric
C_{SK}	spatial skew of MBR
Histograms	**Description**
MinSkew	minSkew, fixed grid
MinSkewProg	minSkew, prog. refinement
rkHist	rK-Hist with $\alpha = 0.1$
R-tree	fixed sized partitioning, Hilbert Curve
R-V	C_V, Hilbert Curve
R-VQP	C_{QP}, Hilbert Curve
R-RK	C_{RK}, Hilbert Curve
R-SK	C_{SK}, Hilbert Curve
FST	STHist forest

Table 1: Studied Methods

5.2 Studied Methods

In our experiments, we study the performance of different histograms. As a reference method we used MinSkew. Histograms produced by MinSkew perform well [7, 1, 16]. We implemented both MinSkew with fixed grid and *progressive refinement* strategy respectively, as described in [1]. We refer to the first as MinSkew and the second as MinSkewProg. For each data and query set we always used the best parameter setting for the grid size. For d=2, we used a grid with 2^{14} cells. This was the the best setting according to accuracy and build-up time in our experiments. For a MinSkewProg we used four grids with $2^{14}, 2^{12}, 2^{10}, 2^8$ cells; again this was the best setting. For d=3, we used 2^{15} cells for MinSkew and four grids with $2^{15}, 2^{12}, 2^9, 2^6$ cells for MinSkewProg. Other examined methods are listed in Table 1.

5.2.1 R-tree Methods

Methods with prefix R (R-V, R-VQP, R-RK, R-SK) are derived from R-trees and our Sort-Partition algorithms. For R-tree methods we set $B = 100$ and $b = 40$ for d = 2 and $B = 72$ and $b = 28$ for d = 3. Recall that B denotes the leaf capacity and b the minimum leaf occupation. Leaf nodes are generated using *gopt** algorithm. In general, this results in more buckets than m (the desired number of buckets). In a second step, we apply *opt** to the leaf bounding boxes to yield exactly m buckets. The chunk size was set to 20'000 rectangles; larger chunk sizes did not yield significantly better histograms. The methods R-V, R-VQP, R-RK, R-SK only differ in their cost function used in *gopt** and *opt** (see Table 1). For example, C_V refers to the cost function minimizing the volume of the bounding boxes. Additionally, we

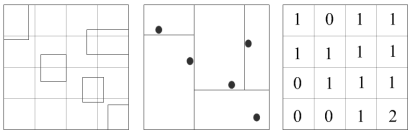

Figure 3: k-Uniformity metric and spatial skew of MBR

implemented rKHist as described in [5] using an underflow rate with $\alpha = 0.1$ (again this was the best setting in our experiments).

We also studied the quality of other cost functions (R-RK, R-SK). C_{RK} is a k-Uniformity metric proposed in [5]. C_{SK} minimizes the skew within a bucket. For a detailed description, see [5, 1]. Figure 3 illustrates how these functions are computed. The left figure shows a bucket region with five rectangles.

The k-Uniformity function C_{RK} is based on a rectangular subdivision of a bucket region. The last is built using the associated point objects [5]. For rectangles we only considered their centers. This subdivision is computed in kd-tree manner. This representation is constructed using the recursive binary splits of a point set. Each dimension of the rectangular bucket split into two in a round-robin fashion. The median is used as the spit point, see center plot of Figure 3. The C_{RK} returns the standard deviation of areas of the resulting rectangles. Note that, the processing cost for a bucket with n elements is $O(n \log n)$. This is a drawback in comparison to other cost functions discussed above.

The function C_{SK} is based on a regular grid[1]. First, the regular grid is computed for a bucket region (see left side of Figure 3). Then, the frequency of objects intersecting a cell is computed for each cell (see right side of Figure 3). The function then returns the standard squared error (SSE) of frequencies. The drawback of the last method is that the grid resolution has to be set as an additional parameter.

We also implemented STHist method [16]. We call it FST, because we used the so-called *forest*-strategy. The build-up cost for FST is very high, particularly for data sets with skewed data distributions. For example, the build-up time was a factor 100 higher for the *rea* data set than for other methods, due to its $O(n^2)$ runtime. In order to conduct all the experiments for FST, we applied FST to a random sample of 10%.

5.2.2 Space Allocation

Recall that each bucket h_i maintains the MBR and the following statistical values: $n_i, dx^i_{avg}, dy^i_{avg}, s_i$. Let w be a size of machine word in bytes. Thus, storage amount of a bucket is $d \cdot w + 2 \cdot w + d \cdot 2 \cdot w$ ($d \cdot w$ bytes are average length information, $2 \cdot w$ byte for number of objects and spatial density and $d \cdot 2 \cdot w$ for MBR). This storage scheme is also used in the original MinSkew approach [1]. It is possible to save storage for MinSkew bucket MBR. Because the MinSkew histogram can be stored as a kd-tree. We assume that the leaf node of kd-tree stores statistical information. We implemented MinSkew using a grid with resolution of power 2. For 2^{dk} cells we need $\log k$ bits to decode the split position. Additionally we store information about the split dimension $\lceil \log d \rceil$ bits and one bit to decode whether a node is a leaf [6].

Thus, for d=2 and d=3, the MinSkew Histogram can keep almost twice as many buckets as R-tree histograms. This is reflected in our experiments. If an R-tree histogram consists of m buckets, we allow MinSkew to use $2 \cdot m$ buckets. In our experiments, space allocation is expressed by the number of buckets m for R-tree Histogram.

Method	Time ms	std.
MinSkew	34089	6781
MinSkewProg	24648	3504
rKHist	23950	2408
R-V	27434	685
FST	41321	56370

Table 2: Build time d=2 data sets for 1'000 buckets

All methods are implemented within the XXL-java library [1]. The experiments are conducted with a 64 bit Intel i7-2600 (2 x 3.4 Ghz), 8 Gb memory machine running Windows 7. For external sort we used 10 MB memory buffer. We examined histograms with $m = 500, 1000, 2000, 3000, 4000$ and $m = 5000$ buckets. Note that previous experiments considered only a small number of buckets. Due to large main memories available, we see the necessity to investigate large histograms.

5.2.3 Error Metrics

Performance quality of the proposed methods was evaluated using different error metrics. We use workload error E_w as default metric. E_w is defined as follows:

$$E_w = \sum_i |act_i - est_i| \,/\, \sum_i act_i$$

Here, act_i is the actual number of answers of the i-th query, and est_i is the estimated number. Note that this measure is commonly used in other experiments [1]. We also considered the average absolute error $E_{abs} = |act_i - est_i|$ and the average relative error $E_{rel} = \frac{|act_i - est_i|}{\max(1, act_i)}$ (as in [16]). They are considered for workloads derived from models M_3 and M_4, because all queries offer the same selectivity.

5.2.4 Build and Estimation Time

All methods except FST were able to build a 1'000 bucket histogram for 1'000'000 data objects for all data sets in less than 1 minute. Average build time in milliseconds is given in table 2 for d=2 data sets for 1'000 bucket histograms. The cardinality of the data sets was limited to 1'000'000 objects. The column std. shows the standard deviation. In general, the rKHist and R-V method are less sensitive to a data distribution compared to MinSkew and MinSkewProg counterparts. Recall that we construct FST histogram using random sampling of 10%. The FST method is very sensitive to data distribution, especially for non-uniform data sets. The build time for a rKhist and R-V method was dominated by external sort. The MinSkew and MinSkewProg were CPU dominated.

The estimation time may become an issue if the number of histogram buckets is too high. The resulting histogram in a simple variant is represented as an array of buckets. To decrease the estimation time, histograms can be represented as main memory R-trees. Figure 4 depicts the function of bucket size and total workload time for different representa-

[1] http://xxl.googlecode.com

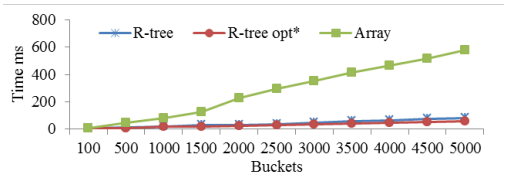

Figure 4: Total estimation time as a function of histogram size for the California data and M_4 query set

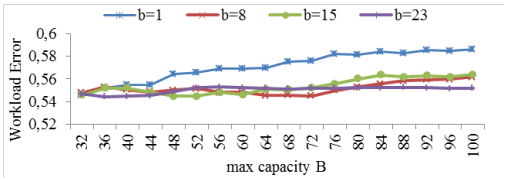

Figure 5: Function of E_w and capacity parameter $b = \{1, 8, 15, 23\}$ and B for the Histogram-Generation Step (California data, M_4)

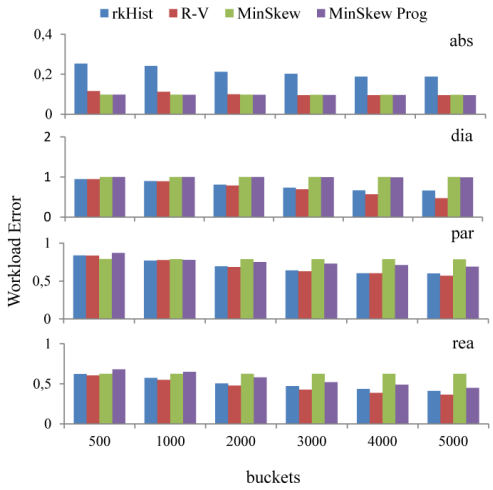

Figure 6: E_w for rectangular data and query set M_4

tion of a histogram. The first two are R-trees. The third one is an array of buckets. For R-tree, we used main memory setting and set the fan-out to 12 entries per node (again it was the best setting in our experiments). Additionally, we build an R-tree using histogram buckets with a *opt** partitioning method and C_V as a cost function. We constructed histograms on California data set and measured the overall estimation time of 10'000 queries from M_4 query set with selectivity 100. We observed that if the bucket number exceeds 100, the R-tree organization displays better results.

5.2.5 *Impact of Bucket Capacity Parameter*

The bucket capacity parameters $b = b = \max(\lfloor N_1/2m \rfloor, 1)$, $B = \lceil N_1/m \rceil + b$ in Histogram-Generation step are set depending on desired bucket number m. We examine parameter sensitivity to show that this setting displays a good accuracy. For each data set we run *opt** with different bucket capacities. Figure 5 shows four *opt** configurations applied on leaf nodes of the California data set generated after the Micro-Clustering. There are $N_1 = 30'398$ leaf nodes generated from 1'888'012 rectangles. We then generate histograms for $m = 1'000$. On average, histogram buckets have capacities about $\lfloor N_1/m \rfloor = 30$ leafs. For each fixed b, we computed a E_w under query model M_4 with selectivity 100 as a function of parameter B. The values $b = 15, B = 46$ exhibit a good performance. We observed that increasing the parameter B does not lead to better histograms especially for a non-uniform data sets. In general, high $B - b$ values do not significantly improve histogram quality and even increase time complexity. In contrast, small $B - b$ values exhibit poor results for uniform and near-uniform data distributions.

5.3 Experimental Results

In this section we present a detailed discussion about accuracies of different histograms. First we describe general trends observed in our experiments. Further, we discuss results obtained for small sized queries (d=2,3). We focus on M_4 workload. Subsequently, we report results for large sized queries. For the sake of brevity, we only present results of rKHist, R-V, MinSkew and MinSkewProg. Other method accuracies are presented if necessary.

We observed several trends: first, although R-tree methods are build based on a M_1 model, they exhibit also good estimation results for other query workloads.

Second, R-tree based methods yield better accuracies for non-uniform data distributions than MinSkew and MinSkewProg for all data and query workloads. Their selectivity accuracies increase more significantly with an increasing number of buckets than for MinSkew and MinsSkewProg. We also observed that with increased number of buckets, the quality of MinSkew improves marginally (as reported in [1]). In contrast, the quality of MinSkewProg increases more significantly. Using several different grid resolutions prevents MinSkewProg from allocating many buckets in a single highly skewed cluster, since the number of buckets produced per grid is equally balanced[1]. For a large number of buckets, MinSkewProg is the better choice than MinSkew.

Third, the general deficiency of R-tree methods for uniform and near-uniform data distribution is corrected using our proposed partitioning methods. This can be explained by the fact that the produced MBRs display almost no overlap, thus, this partitioning minimizes the estimation error.

5.3.1 *Workload M_4*

In this section, we present result for query model M_4 (query follow data distribution and query size is expressed by the number of results). Since the workload M_4 is more realistic and more difficult to handle, we report results for this model. Results for other models are discussed if necessary.

Figure 6 and 7 show results of rKhist, R-V, MinSkew and MinSkewProg for a d=2 data sets and query workload M_4 with selectivity 100. We bundle results for rectangular data sets in Figure 6 and for point data in Figure 7. Best results are achieved on *ped* data set. This data set consists of thin shaped clusters of points. Minimizing the MBR volume using dynamic programming scheme leads also to a thin shapes of MBRs, thus minimizing the estimation error. The rKHist method as well as the simple R-tree method (fixed size partitioning) have problems with uniform and near-uniform data sets. The rKHist greedy split strategy does not lead to

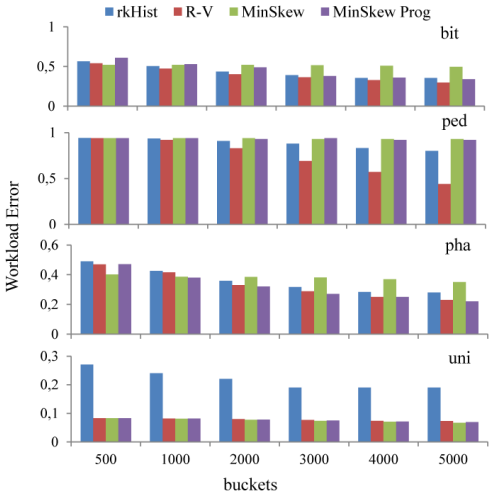

Figure 7: E_w **for point data and query set** M_4

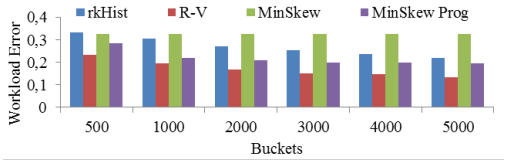

Figure 8: E_w **for d=3 data set *rea* and query set** M_4

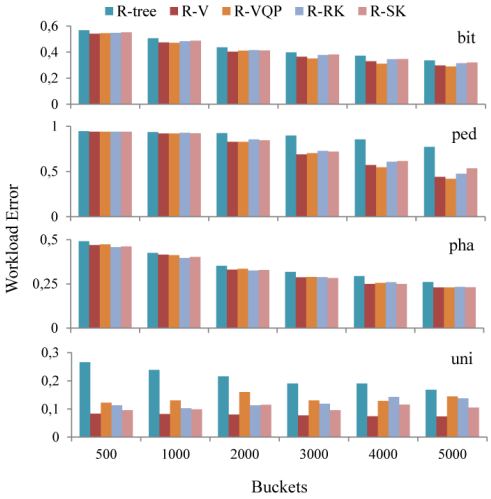

Figure 9: E_w **for point data and query set** M_4

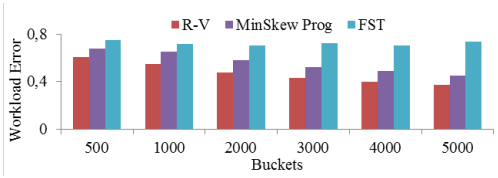

Figure 10: E_w **for d=2 data set *rea* and query model** M_4

a partitioning with small overlap introducing high estimation error. In contrast, R-V method yields better partitioning and its accuracy is comparable with MinSkew and MinSkewProg accuracies. R-V and rKHist perform better for non-uniform data sets *bit, dia, par, ped* and *rea* than MinSkew and MinSkewProg with increasing number of buckets. For *par* data set, we observed almost no difference between rKHist and R-V method. This data set has a high variance in shapes and sizes of rectangles and is difficult to handle either by R-tree histogram and index.

Figure 9 depicts results of R-tree methods compared with a fixed size partitioning strategy (R-tree) for d=2 point data. In general, we observed that all methods using our optimized sort-partition framework display better accuracy than R-tree. Estimation accuracies of R-V, R-VQP, R-RK and R-SK do not differ significantly for non-uniform data distributions. However, C_V function exhibit better results for uniform data sets than other cost functions.

For d=3 we obtain similar results as for d=2 for all data and query sets. In general, estimation quality are slightly better for non-uniform data sets than for d=2. Figure 8 reports results for d=3 *rea* data set.

In Figure 10, we report the E_w for FST method compared with R-V and MinSkewProg for *rea* data set. FST Performance was very poor for all data and query sets, as we used random sampling for input data. Although applying this method on whole data set does not display better results than rKHist, R-V and MinSkewProg methods.

5.3.2 *Results for Large Queries*

Figure 11 shows results for the California (*rea*) data set for all query workloads. For large queries R-tree based methods yield even better accuracy then MinSkew counterparts in comparison with small sized queries. Similar to small sized queries best result are achieved for non-uniform data sets. rKHist performs for two uniform *uni, abs* sets very poor in comparison with R-V, MinSkew and MinSkewProg. Although for large queries on the California data set accuracy difference between rKHist and R-V was not that significant, with a high number of buckets R-V method was superior to other methods. Best results for R-V we achieve for synthetic data sets*abs, bit, dia, ped, pha.* Again results for *par* data set are comparable with a small sized query results. One possible solution is to partition such data distribution according to the object size and shape and construct histograms or index for each partition independently.

6. CONCLUSIONS

Spatial histograms are becoming increasingly important for modern GIS applications. They provide a first inexpensive view on large spatial data sets; and therefore are ideally suited for visualization and approximate query processing. In this paper, we introduce a novel histogram method derived from a bulk-loading algorithm of R-trees. It largely eliminates the cumbersome need for setting parameters; the only ones (page capacity B and minimum occupation b) are set in the same manner as it is known for R-trees. In general, our histogram method is fairly easy to implement because it combines elementary building blocks like sorting and dynamic programming. Our method also overcomes the weak performance of R-tree histograms in case of uniformly dis-

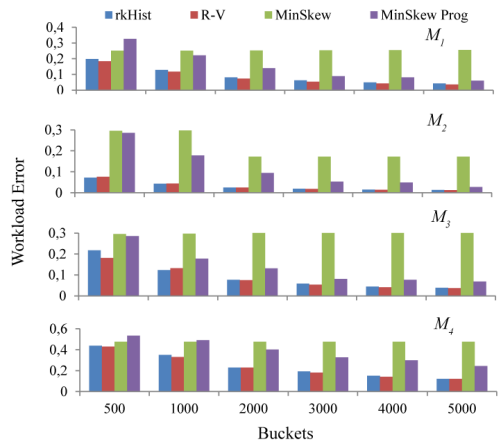

Figure 11: E_w for the California data set; M_1, M_2 with volume 0.1 and M_3, M_4 with selectivity 1000

tributed records. Until now, it has been considered to be an open problem whether accurate R-tree histograms can be developed for uniformly distributed data. For real data sets that are known to be highly non-uniform our method generates histograms of high quality, generally much better than the ones generated by other methods.

This paper also introduces a new kind of experimental setup for spatial histograms. Inspired by cost models for spatial indexes, we consider different kind of workload scenarios rather than putting the focus only on uniformly distributed queries. This gives a more meaningful interpretation of the advantages and disadvantages of spatial histograms. In addition, we also examine the performance of histograms with a rather large number of buckets. Despite the fact of the availability of large main memories, there have been only a very few results available for histograms with more than 1000 buckets. In fact, our experiments reveal that not all of the state-of-the-art histograms can improve quality with an increasing number of buckets.

While our focus is on two- and three-dimensional data, we are currently interested in the design of histograms for high-dimensional data. Similar to our design of accurate spatial histograms, we expect that the design principles of high-dimensional indexing can be reused again for high-dimensional histograms.

7. ACKNOWLEDGMENTS

The authors would like to thank Eugen Walter, Mareike Stoof and Anne Sophie Knöller for help in implementation and reviewing this piece of work.

8. REFERENCES

[1] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *SIGMOD '99*, pages 13–24, New York, NY, USA, 1999. ACM.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB '03*, pages 81–92. VLDB Endowment, 2003.

[3] N. Beckmann and B. Seeger. A revised r*-tree in comparison with related index structures. In *SIGMOD '09*, pages 799–812, New York, NY, USA, 2009. ACM.

[4] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: a multidimensional workload-aware histogram. *SIGMOD Rec.*, 30:211–222, May 2001.

[5] T. Eavis and A. Lopez. Rk-hist: an r-tree based histogram for multi-dimensional selectivity estimation. In *CIKM '07*, pages 475–484, New York, NY, USA, 2007. ACM.

[6] F. Furfaro, G. M. Mazzeo, D. Saccà, and C. Sirangelo. Hierarchical binary histograms for summarizing multi-dimensional data. In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, pages 598–603, New York, NY, USA, 2005. ACM.

[7] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. *SIGMOD Rec.*, 29:463–474, May 2000.

[8] Y. Ioannidis. The history of histograms (abridged). In *VLDB '2003*, pages 19–30. VLDB Endowment, 2003.

[9] H. V. Jagadish, V. Poosala, N. Koudas, K. Sevcik, S. Muthukrishnan, and T. Suel. Optimal histograms with quality guarantees. In *In VLDB*, pages 275–286, 1998.

[10] I. Kamel and C. Faloutsos. On packing r-trees. In *CIKM '93*, pages 490–499, New York, NY, USA, 1993. ACM.

[11] M. Muralikrishna and D. J. DeWitt. Equi-depth multidimensional histograms. *SIGMOD Rec.*, 17:28–36, June 1988.

[12] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In *ICDT '99*, pages 236–256, London, UK, 1999. Springer-Verlag.

[13] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *PODS '84*, pages 181–190, New York, NY, USA, 1984. ACM.

[14] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *PODS '93*, pages 214–221, New York, NY, USA, 1993. ACM.

[15] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. *SIGMOD Rec.*, 25:294–305, June 1996.

[16] Y. J. Roh, J. H. Kim, Y. D. Chung, J. H. Son, and M. H. Kim. Hierarchically organized skew-tolerant histograms for geographic data objects. In *SIGMOD '10*, pages 627–638, New York, NY, USA, 2010. ACM.

[17] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed r-trees. In *SIGMOD Conference*, pages 17–31, 1985.

[18] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. Isomer: Consistent histogram construction using query feedback. In *ICDE '06*, pages 39–, Washington, DC, USA, 2006. IEEE Computer Society.

[19] Y. Theodoridis and T. Sellis. A model for the prediction of r-tree performance. In *PODS '96*, pages 161–171, New York, NY, USA, 1996. ACM.

[20] K. Yi, X. Lian, F. Li, and L. Chen. The world in a nutshell: Concise range queries. *IEEE Trans. Knowl. Data Eng.*, 23(1):139–154, 2011.