Hierarchically Organized Skew-Tolerant Histograms for Geographic Data Objects

Yohan J. Roh
SAIT, Samsung Electronics
yohan.roh@samsung.com

Jae Ho Kim
KAIST
jaeho@dbserver.kaist.ac.kr

Yon Dohn Chung
Korea University
ydchung@korea.ac.kr

Jin Hyun Son
Hanyang University
jhson@hanyang.ac.kr

Myoung Ho Kim
KAIST
mhkim@dbserver.kaist.ac.kr

ABSTRACT

Histograms have been widely used for fast estimation of query result sizes in query optimization. In this paper, we propose a new histogram method, called the Skew-Tolerant Histogram (STHistogram) for two or three dimensional geographic data objects that are used in many real-world applications in practice. The proposed method provides a significantly enhanced accuracy in a robust manner even for the data set that has a highly skewed distribution. Our method detects hotspots present in various parts of a data set and exploits them in organizing histogram buckets. For this purpose, we first define the concept of a hotspot, and provide an algorithm that efficiently extracts hotspots from the given data set. Then, we present our histogram construction method that utilizes hotspot information. We also describe how to estimate query result sizes by using the proposed histogram. We show through extensive performance experiments that the proposed method provides better performance than other existing methods.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications – *spatial databases and GIS*

General Terms

Algorithms, Experimentation, Performance

Keywords

Spatial Databases, Query Optimization, Histograms

1. INTRODUCTION

Histograms have been widely used as an approximation tool for multi-dimensional data distributions. The most important application of histograms is to estimate the result sizes of queries. This so-called selectivity estimation is used by query optimizers to determine the most efficient query execution plan [14, 15]. In addition, selectivity estimation has been shown to be useful in many other areas of database processing, e.g., top-*k* query processing, skyline query processing, load-balancing in parallel join query execution, and spatio-temporal query processing [3, 6, 21, 22, 26, 27]. Motivated by such applications, there has been a great deal of work on the problem of selectivity estimation such as histograms [1, 2, 8, 11, 12, 19, 24, 28], wavelet transformation [18, 29], SVD [23], discrete cosine transform [17], and sampling [13]. Among these approaches, histograms have been shown to be one of the most popular and effective ways to obtain accurate estimates of selectivity for multi-dimensional queries [8]. A histogram consists of a set of buckets B_i, $i = 1, \ldots, n$, where each bucket B_i has its data space S_i and frequency F_i of data objects within S_i. The number of buckets for a histogram is usually a system parameter. Given a data range I specified in a query, an estimate of the selectivity of the query, i.e., the number of objects in I, is computed as follows, under *uniform distribution assumption*: $\sum_{i=1\ldots n}(|S_i \wedge I|/|S_i| \cdot F_i)$. Here, | | denotes the size of data space and '$S_i \wedge I$' denotes the intersection of S_i and I.

EXAMPLE 1. *Consider a histogram that consists of three buckets* B_1 (S: 0~50, F: 100), B_2 (S: 50~80, F: 40), *and* B_3 (S: 80~100, F: 60) *for one-dimensional data domain* [0 .. 100]. *Suppose a query range is* [30 .. 90]. *Then, the selectivity estimate for this query is computed as follows*: 20/50·100 + 30/30·40 + 10/20·60 = 110.

In this paper, we explore how to construct an effective histogram for selectivity estimates of range queries. We will focus on the histograms for two or three dimensional geographic objects. Let us consider cases of two-dimensional geographic data. Rectangle-shaped regions are commonly used for the data spaces of buckets. Each bucket has the specification of its data space or region together with the frequency of data objects (i.e., total number of data objects within it). Suppose that all the data objects in the region of a bucket are uniformly distributed. When a query is given, an estimated selectivity value for one bucket is computed in proportion to the size of the overlapping region between the query region and the bucket's region. The selectivity estimate for a query is the sum of all the estimated values for all the buckets as in Example 1.

Skewness Problem. Consider Example 1 again. While the region of bucket B_2 is fully contained in the query range, the regions of buckets B_1 and B_3 partially overlap with the query range. (From now on, for convenience, we will often use the term "bucket" to denote the region of a bucket, i.e., data space associated with the bucket, if there is no ambiguity.) In estimating a selectivity of the

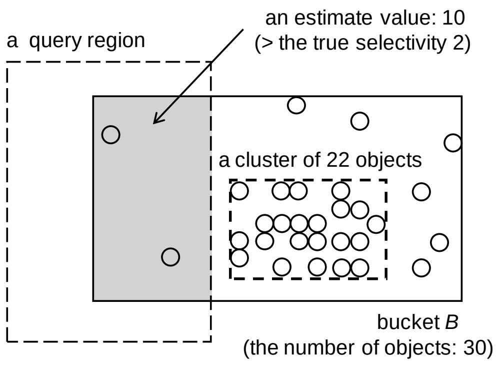


Figure 1: A bucket containing a cluster of objects.

query, we do not have a problem for buckets fully contained in the query range. However, buckets that partially overlap with the query range may cause a problem. In the example, we have computed the selectivity estimate under uniform distribution of data in each bucket B_1 and B_3. However, if data distribution is highly skewed in buckets B_1 and B_3, the selectivity estimate computed in Example 1 can be significantly deviated from the actual frequency of objects. Now, as an example of two dimensional data space, consider a certain bucket B in a two dimensional histogram that overlaps with a given query region as shown in Figure 1. In bucket B, 22 objects out of 30 objects are densely clustered, and this cluster of objects is outside the query region. If selectivity estimation is made based on the uniform distribution assumption, this cluster will cause an estimated selectivity to deviate significantly from the actual frequency of objects. Suppose the size of the overlapping region is one-third of the size of bucket B. Then the estimated number of objects for this overlapping region is $(1/3)\cdot 30 = 10$, which is five times larger than the actual frequency 2. Note that this problem stems from the inappropriate organization of buckets in constructing a histogram. In other words, a histogram should not have a bucket within which there is a heavy cluster of objects such as bucket B in Figure 1.

Many existing histograms do not cope with such clusters well. In this paper, we propose a new histogram method, called the Skew-Tolerant Histogram (STHistogram). When constructing a histogram, the STHistogram detects and utilizes the clusters of objects in data space. By directly utilizing clusters in organizing buckets, our proposed method can provide an enhanced accuracy in a robust manner over skewed distributions. Through extensive performance experiments, we show a significant accuracy improvement of the proposed method and its robustness to skewed data distribution. Note that, in the histogram for the one dimensional data space as in Example 1, the number of buckets that partially overlap with the given query range is usually two. But in the histogrtam for two or three dimensional geographic space, the number of buckets that partially overlap with the given query region is much larger than two in general.

The rest of the paper is organized as follows. Section 2 describes related work. In Section 3, we present most of main ideas in this work and propose our basic histogram method. Section 4 extends the basic method proposed in Section 3 to cover the cases when the basic method only may not work effectively. Section 5 provides the results of performance experiments with six real-life data sets as well as one synthetic data set. We draw our conclusions in Section 6.

2. RELATED WORK

Histograms on multiple attributes of a relation have been widely used for various types of query processing, e.g., query optimization, top-*k* query processing, skyline query processing, load-balancing, spatio-temporal query processing. For query optimization, histograms are popularly used in commercial database systems to estimate the result sizes of (sub)queries and to develop the most efficient query execution plans [14, 15]. For top-*k* query processing, Bruno et al. [3] and Chaudhuri et al. [5] utilize histograms for translating a top-*k* request into a single range query that can be efficiently processed by existing database engines. They have shown that the use of histograms can avoid the requirement of a full sequential scans of the database and significantly reduce the time required to support top-*k* queries. For skyline query processing, Chaudhuri et al. [4] and Papadias et al. [21] use the statistics of histograms, such as *MinSkew* [1], to accurately estimate the result sizes of skyline queries. These estimated values are shown to be useful for providing immediate feedback to users and implementing skyline computation as an operator within database systems. For load-balancing of parallel hash joins, Poosala and Ioannidis [22] make use of histograms to accurately estimate the cost required to perform the join operation, and effectively balance the load across nodes that participate in the parallel execution. For spatio-temporal query processing, the authors of [6] use the *MinSkew* histogram [1] and extend it with velocities to estimate the selectivity of spatio-temporal window queries, i.e., the number of objects that will appear in the query window at a given future time. For the same purpose, Tao et al. [27] propose histogram-based solutions to effectively deal with the dynamics of the moving objects. Sun et al. [26] use the *MinSkew* histogram [1] to accurately estimate the selectivity of spatio-temporal joins, i.e., given two sets S_1 and S_2 of objects, the number of pairs $<o_1, o_2>$ of objects such that $o_1 \in S_1$, $o_2 \in S_2$, and the distance between the two objects at a given future time is below a certain threshold.

So far, many studies have been carried out with different approaches in order to enhance the accuracy of multi-dimensional histograms. The basic assumption in using a multi-dimensional histogram is that the histogram works well when data is uniformly distributed in every bucket. However, the problem of organizing buckets in such a way that the data is uniformly distributed in every bucket has been shown to be NP-hard [20]. Therefore, most existing methods use their own heuristics as follows.

The *EquiDepth* histogram method [19] partitions the data space, one dimension at a time. Here, in each *i*-th dimension, the data space is divided into v_i intervals, each of which has the same number of data objects. So, for a *d*-dimensional data set, a set of $v_1 \times v_2 \times \ldots \times v_d$ buckets is constructed, where each bucket contains the same number of data objects. The *EquiDepth* histogram may be faster to construct among other types of multi-dimensional histograms. But, because of its rigid structure, it may be not flexible to cope with various cases of data skew.

The *MinSkew* histogram method [1] initially divides the data space into a uniform grid of rectangular regions. Then, it performs repeatedly *binary space partitioning*, where a bucket is partitioned into two sub-buckets. This partitioning approach may construct histograms rapidly; however, it may not recognize multi-dimensional subregions where data are not uniformly distributed, which may degrade the accuracy of histograms. This is because the partitioning heuristics of *MinSkew* is based on data skew in only one-dimension at a time rather than considering the skew of multiple dimensions at once. In this method, the number of grid cells is a user-provided parameter. The estimation accuracy of *MinSkew* may vary depending on this parameter [1]. In practice, it is difficult for users to provide the optimal or a near optimal value for the required parameter.

The *GenHist* histogram method [11, 12] uses *multi-dimensional grids* of various sizes. In this method, high-frequency grid cells are converted into buckets. More specifically, it iteratively constructs a certain number of buckets by using grids. Being different from the aforementioned approaches, this method directly approximates multi-dimensional (i.e., joint) data distributions. The authors of *GenHist* claim that the *GenHist* histogram behaves more accurately in high dimension than the previous approaches on selectivity estimation for multi-dimensional queries, such as random sampling, *Wavelet* [29], *EquiDepth* [19], and *MinSkew* [1].

The *RK-Hist* histogram method, which has been recently proposed in [8], uses a special type of an r-tree index, i.e., *Hilbert packed R-tree* [16], where the entire data objects are sorted based on their positions along the Hilbert curve. The sorted objects are divided into several (leaf) nodes, in which the size of each leaf node is a disk block. Then, the *RK-Hist* method creates an initial set of buckets, where each bucket is constructed by merging a fixed number of leaf nodes. For each initial bucket, the skew of data is computed and then some bucket with a high skew is split into two in a repeatable manner, until the total number of buckets becomes the predefined number or there is no improvement of the total skew of data in buckets. The authors of *RK-Hist* claim that the *RK-Hist* histogram outperforms other existing methods, such as *EquiDepth* [19] and *GenHist* [11, 12] in terms of accuracy. However, the *RK-Hist* method may introduce unnecessary buckets, when a fixed number (say *p*) of leaf nodes is merged into an initial bucket. For example, consider a node *u* with a very low skew that is an ancestor of a large number of leaf nodes. If the number of the descendant leaf nodes is much greater than *p*, several buckets will be constructed from these leaf nodes. But, only one bucket consisting of a single node *u* suffices to provide accurate estimation of selectivity (instead of several buckets).

There are several approaches for bucket layout. In the *grid approach*, buckets are arranged in rows and columns, e.g., as in the well-known equal-width histogram. In the *recursively partitioning approach*, an existing bucket is recursively divided into two sub-buckets along some dimension, e.g., as in *MinSkew* [1]. There are also other approaches for bucket layout that impose fewer restrictions than the aforementioned approaches on the arrangement of buckets, i.e., allow a newly created bucket to cover a portion of data space in a more flexible way. For example, in the methods of *GenHist* [11, 12] and *RK-Hist* [8], the regions of buckets are allowed to overlap. There can be a *nested approach*, where the region of any bucket, except the bucket whose region covers the whole data space, is fully contained in the region of some other bucket. Then, histograms from the nested construction of buckets can be represented as hierarchies of buckets. Our histogram proposed in this paper has nested layout of buckets.

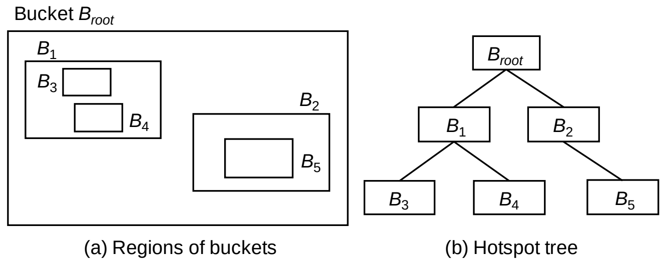


Figure 2: An example histogram constructed by the proposed STHistogram.

Histograms are typically recomputed to reflect updates of the underlying data in a periodic manner. There is another interesting approach to maintaining histograms, called the *self-tuning histogram* [2, 24]. This approach incrementally maintains buckets in response to feedback from the query execution engine about the actual cardinalities of range queries. Because these cardinalities reflect data updates, the approach can gracefully adapt buckets to the updates of the underlying data set. But there are following limitations in this approach. First, because only the regions of queries that have been processed are used, only buckets related to those queries can be updated. That is, updates in the other regions may not be reflected. Second, the feedback-based maintenance inevitably incurs additional overhead on the query processing [2, 24]. There is other research on maintenance of histograms [9, 28].

3. THE PROPOSED METHOD

We present in this section our new histogram method, called the Skew-Tolerant Histogram (STHistogram). We focus on the histograms for two or three dimensional data used in geographic information processing. In what follows, we will mainly use a rectangular shaped two dimensional data space for simplicity of our discussion.

In the proposed method, rectangular shaped cluster regions, called “hotspots” are detected and exploited for organizing buckets hierarchically. The use of hotspots in constructing histograms can effectively reduce the accuracy degradation caused by skewness mentioned in Section 1.

3.1 Sketch of the Proposed Method

We first describe a sketch of our proposed method that shows the idea of a histogram constructed by STHistogram. In the proposed method, hotspots in the data space are recursively found and made as buckets, which forms a hierarchy of buckets called a *hotspot tree*.

Figure 2 illustrates construction of a hotspot tree. Suppose a data set and the total number of buckets for a histogram are given. Initially, bucket B_{root} that is the minimum bounding rectangle for a whole data set is made as the root of a hotspot tree. Next, within bucket B_{root}, we find hotspots and organize them as children of B_{root} in the hotspot tree, i.e., buckets B_1 and B_2 in the figure. The number of children of B_{root}, “two” in this case, depends on both the distribution of a data set and conditions for a certain region to be a hotspot. The similar process continues until the total number of buckets in the hotspot tree becomes the predefined number of buckets for the histogram. How to find hotspots within a bucket, conditions for a certain region to be a hotspot and the details of how to organize a hotspot tree will be discussed in the following sections. Note that a hotspot tree is our target histogram, so a node in the hotspot tree and a bucket will be used interchangeably. Note also that all the buckets, except B_{root}, are nested within other buckets in the higher levels of the hotspot tree.

3.2 The Concept of a Hotspot

A hotspot in two dimensional data space is a rectangular region where objects are closely located together with a high density, and satisfies certain conditions described in Definition 2. Let the "density" be "frequency/size", as usual.

DEFINITION 1 (RELATIVE DENSITY OF A REGION). *For a subregion R in a bucket B, the relative density of R with respect to bucket B, denoted by* $RelativeDensity_B(R)$*, is defined as*

$$RelativeDensity_B(R) = density(R)/density(B).$$

Informally, a hotspot R is the subregion whose density is at least k times greater than the density of the enclosing bucket B, for some k. The value k is the minimum value of the relative density of region R to the enclosing bucket B so that region R can be a hotspot.

EXAMPLE 2. *Let R be a subregion in a bucket B. Suppose that we use* $k = 2$. *Suppose also that* $size(R) = (1/4)\cdot size(B)$ *and the number of objects in R is the half of the number of objects in B. Then,* $RelativeDensity_B(R) = 2$, *i.e., the density of R is two times greater than that of B, and hence R is a hotspot*.

When bucket B is the nearest enclosing bucket of subregion R (e.g., in Figure 2 the nearest enclosing bucket of bucket B_3 is bucket B_1), we will use $RelativeDensity(R)$ to denote $RelativeDensity_B(R)$. That is, we omit "B" in $RelativeDensity_B(R)$ if bucket B is the nearest enclosing bucket of region R.

In the following, we use two parameters f and s for definition of a hotspot. Basically, determination of a hotspot is based on the density of a region, which also depends on the size of a region and the frequency of objects in that region. Thus, in the following definition, instead of simply using the density parameter k as in Example 2, we use the size parameter s and the frequency parameter f. For a region R, let $freq(R)$ denote the number of objects in R. $freq(R)$ will be called "the object frequency of R".

DEFINITION 2 (HOTSPOT).

(1) *Suppose there is a bucket B whose size and object frequency are S and F, respectively. Given two variables s and f such that* $s > 1$ *and* $s \geq f$, *a rectangular subregion R of bucket B that satisfies the following conditions* (1.1), (1.2), *and* (1.3) *is a hotspot.*

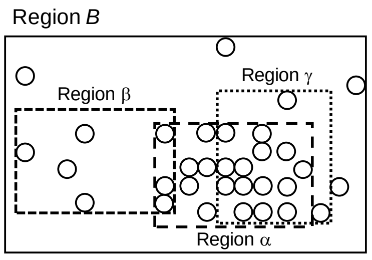
(1.1) $size(R) \leq S/s$.

(1.2) $freq(R) \geq F/f$.

(1.3) *The shape of R is the same as that of B.*

(2) *Any two hotspots in the same nearest enclosing bucket, i.e., any two sibling nodes in a hotspot tree, are mutually exclusive.*

In the definition of a hotspot, condition (1.1) is called "size-condition", (1.2) is called "frequency-condition", and (1.3) is called "shape-condition". Here, the definition of "the same shape" in condition (1.3) is as follows: A rectangular region can be defined by two points, i.e., a bottom-left point and a top-right point. Consider a rectangular region R_a whose bottom-left and top-right points are (a_{x1}, a_{y1}) and (a_{x2}, a_{y2}), respectively. Consider also a rectangular region R_b defined by two points (b_{x1}, b_{y1}) and (b_{x2}, b_{y2}) in the same manner. We say that the shapes of R_a and R_b are the same if $(a_{y2} - a_{y1})/(a_{x2} - a_{x1}) = (b_{y2} - b_{y1})/(b_{x2} - b_{x1})$. Condition (2), i.e., the last part of the definition, stipulates that a set of hotspots in the same nearest enclosing bucket must not overlap with one another.



(a) Region and subregions

Region	Object Frequency	Size
B	32	32
α	22	8
β	7	8
γ	16	6

(b) Information about regions

Figure 3: An example of the hotspot.

The variable s in condition (1.1) is used to specify the size (i.e., less than or equal to S/s) of a hotspot. The variable f in condition (1.2) is used to specify the minimum object frequency (i.e., F/f) of a hotspot. For example, $s = 4$ and $f = 2$ says the following condition: In order for a region R to be qualified as a hotspot, the size of R must be at most one-fourth of the size of the enclosing bucket B, and the object frequency of R must be at least one-half of the object frequency of B. In our method, the values of s and f are not user parameters, but are dynamically determined during construction of a hotspot tree, based on the number of hotspots to be detected and the degree of skew in data distribution. This will be explained in Section 3.4. Condition (1.3), i.e., the shape-condition, is only for computational simplicity. This is because the complexity of the problem will be overwhelmingly high if rectangles with all possible shapes are considered.

PROPERTY 1. *For any bucket, the number of hotspots within it is at most* $\lfloor f \rfloor$.

EXAMPLE 3. *Consider three rectangular subregions* α, β, *and* γ *in region B as in Figure* 3(*a*). *The object frequencies and sizes of B and three subregions are described in Figure* 3(*b*). *If we use* $s = 4$ *and* $f = 2$, *subregion* α *is a hotspot while* β *and* γ *are not. Subregion* β *does not satisfy the frequency-condition* (*i.e.*, $7 < 32/2$), *and subregion* γ *violates the shape-condition* (*i.e.*, *the shape of* γ *is not the same as that of B*).

PROPERTY 2. *For a hotspot R,* $RelativeDensity(R)$ *is at least s/f.*

This property is a direct consequence of Definition 1 and Definition 2.

REMARK 1. *In Definition* 2, "$s > f$" *would be better than* "$s \geq f$" *for the general meaning of a hotspot. We include the equality to make our proposed method work well even for the case that data is uniformly distributed. This will be explained in Section* 3.4.

3.3 Detecting Hotspots

Let S and F denote the size and the object frequency of a region D, respectively. s and f are parameters for a hotspot as in Definition 2. To detect a hotspot in region D, we may need to investigate all the subregions of D i) that have the same shape

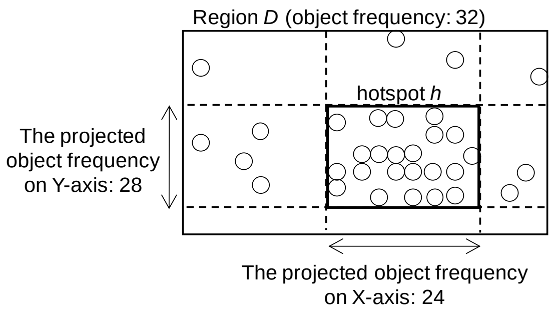

Region *D* (object frequency: 32)
hotspot *h*
The projected object frequency on Y-axis: 28
The projected object frequency on X-axis: 24

Figure 4: An example of the MPF condition.

with *D*, and ii) whose sizes are less than or equal to *S*/*s* by Definition 2. Though any subregion *R* satisfying the above conditions can be a potential candidate for a hotspot, investigating all such subregions is computationally intractable, i.e., the number of such subregions is too much to be investigated. Thus, for computational simplicity, we will use the rectangle of size *S*/*s* that has the same shape with *D* as the basic unit for investigating a hotspot. Besides computational convenience, this policy can prevent too many hotpots from being produced, by clustering several closely located tiny clusters into a single hotspot.

The following is a naive method that searches all subregions satisfying the frequency condition, the shape condition, and the size condition with the size equal to *S*/*s*.

(1) Consider every rectangle *R* that satisfies the following.

(a) The size of *R* is *S*/*s*.

(b) *R* has the same shape as the enclosing region *D*.

(c) The bottom-left corner of *R* is (x_i, y_i) such that x_i is any x-value of an object in region *D* and y_i is any y-value of an object in *D*. Here, the x-value (or y-value) of an object denotes the value of the X-axis (or the Y-axis) of the object.

(2) Check if the number of objects in *R* is greater than or equal to *F*/*f*.

Let *N* be the total number of objects, i.e., the object frequency of a given region. The number of all the possible subregions is N^2, and the examination of each subregion on the *frequency-condition* requires O(*N*) time. Therefore, the complexity of this naive method is O(N^3).

In what follows, we propose an algorithm that identifies hotspots more efficiently. The MPF condition in Property 3 below is useful to reduce the number of possible candidate regions.

DEFINITION 3. *Let R be a hotspot in a region D. Let the bottom-left corner and the top-right corner of R be* (x_i, y_i) *and* (x_j, y_j), *respectively. The projected object frequency of R on the* X*-axis denotes the number of objects in D whose x-values are in* [x_i .. x_j]. *The projected object frequency of R on the* Y*-axis is defined in the same manner.*

PROPERTY 3. (MINIMUM PROJECTED FREQUENCY (MPF) CONDITION). *Let R be a hotspot in a region whose object frequency is F. Then, both of the projected object frequencies of R on the* X*-axis and the* Y*-axis are greater than or equal to the minimum object frequency* (*i.e., F*/*f*) *for the hotspot.*

```
ALGORITHM Detect2DiHotspots(D, O, s, f)
INPUT: D – a two dimensional region (object frequency = F).
    O – a set of two dimensional point objects in D,
        each of which is represented by (x_i, y_i). x_i and y_i are
        values of the X-axis and the Y-axis of the object.
    s – value specifying the size (= S/s) of a hotspot.
    f – value specifying the minimum object frequency (= F/f) of a hotspot.
OUTPUT: SetOfHotspots – a set of hotspots.
1: SortedListX ← Sort objects in O based on the x-values
        in nondecreasing order;
2: SortedListY ← Sort objects in O based on the y-values
        in nondecreasing order;
3: w ← (1/s^{1/2})·the-width-of-D;  /* w is the width of the hotspot */
4: h ← (1/s^{1/2})·the-height-of-D;  /* h is the height of the hotspot */
5: Let o_i represented by (x_i, y_i) be the first object in SortedListX;
6: WHILE (o_i exists) DO {
/*  Let W denote the range [x_i .. (x_i + w)], i.e., a width interval on the X-axis. */
/*  Let freq(W) denote a frequency of objects whose x-values are in W. */
7:    IF (freq(W) ≥ F/f) THEN {
/*  Examine sub-regions whose width is [x_i .. (x_i + w)]. */
8:      SortedCandiListY ← Sort objects in SortedListY based on the y-values
            whose x-values are in W;
9:      Let o_j represented by (x_j, y_j) be the first object in SortedCandiListY;
10:     WHILE (o_j exists) DO {
/*  Let H denote the range [y_j .. (y_j + h)], i.e., a height interval on the Y-axis. */
/*  Let freq(H) denote a frequency of objects in SortedCandiListY
    whose y-values are in H. */
11:       IF (freq(H) ≥ F/f) THEN {
/*  Let a rectangle whose bottom-left and top-right points
    are (x_i, y_j) and (x_i + w, y_j + h) be R. */
12:         R′← AdjustDataSpace(R);
13:         IF (the rectangle R′ does not overlap with any rectangle
                in SetofHotspots) THEN {
14:           Add R′ into SetOfHotspots;
15:           Remove objects inside R′ from SortedListX and SortedListY.
16:           o_j ← the next object in SortedCandiListY whose
                  y-value is greater than (y_j + h);
17:         }
18:       }
19:       ELSE o_j ← the next object in SortedCandiListY;
20:     } // END-OF-WHILE
21:   }
22:   ELSE o_i ← the next object in SortedListX;
23: } // END-OF-WHILE
24: RETURN SetOfHotspots;
```

Figure 5: A hotspot detection algorithm.

EXAMPLE 4. *Suppose that* f = 2 *is the frequency-condition for a hotspot. Let h be a hotspot in region D as shown in Figure* 4. *The object frequencies of h and D are* 22 *and* 32, *respectively. The projected object frequency of h on the* X*-axis and the projected object frequency of h on the* Y*-axis of the hotspot are* 24 *and* 28, *respectively. Each value is greater than the minimum object frequency* (= 32/2) *for the hotspot*.

In Figure 5, we present a hotspot detection algorithm for two dimensional data, called Algorithm Detect2DiHotspots. In this algorithm, we first compute the width and the height of hotspots by using the size- (i.e., *S*/*s*) and shape-conditions. Then, for only intervals of the X-axis whose projected object frequency is greater

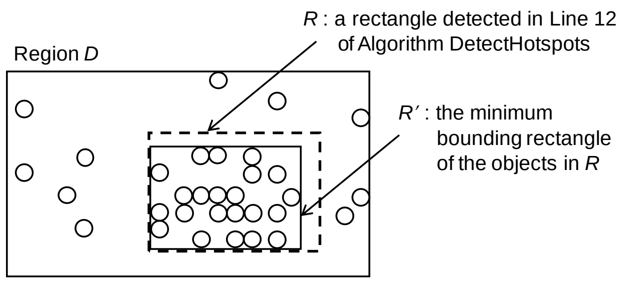

R : a rectangle detected in Line 12 of Algorithm DetectHotspots
Region D
R' : the minimum bounding rectangle of the objects in R

Figure 6: Adjustment of data space for a hotspot.

than or equal to the minimum object frequency (i.e., F/f) for the hotspot, we continue the exploration (Line 7). That is, for each height interval of the Y-axis, check if each candidate region satisfies the frequency-condition for the hotspot (Line 11). In the algorithm, a width (or height) interval is equal to $(1/s^{1/2})$*width-(or height)-of-region-D. This stems from the size-condition (i.e., S/s) and the shape-condition. When a region satisfying the size-, shape-, and frequency-conditions does not overlap with any other hotpots ever detected, it is decided to be a hotspot (Lines 12~14).

AdjustDataSpace in Line 12. *R'* in Line 12 denotes the minimum bounding rectangle of the objects in *R*, as shown in Figure 6. *R* and *R'* may be the same rectangle in some cases, but the size of *R'* will be less than that of *R* in general. Since the objective of Algorithm Detect2DiHotspots is to find a region in which a collection of objects is densely clustered, *R'* captures more accurate data space information for the cluster of objects than *R* does.

Accelerating the algorithm. The proposed algorithm exploits the MPF condition on the X-axis in Property 3 to reduce the search space. In other words, if the MPF condition on the X-axis is not satisfied in Line 7, we skip subregions whose width is $[x_i .. (x_i + w)]$, and advance to the next step in Line 22. We can also utilize the following to make a hotspot detection procedure more efficient:

- **Swift Examination of the MPF condition.** Line 7, which examines the MPF condition on the X-axis, can be performed in a constant time as follows: Let the position of o_i in the *SortedListX* be *p*. The MPF condition is satisfied, if and only if the x-value of the object in *SortedListX*$[p + (F/f) – 1]$ is less than or equal to $x_i + w$. Similarly, Line 11 can be performed in a constant time.
- **Rapid Acquisition of the *SortedCandiListY*.** Line 8 sorts objects based on the values of the Y-axis, whose x-values are in *W*. This step can be performed in O(*N*) by retrieving the objects from the *SortedListY*, whose x-values are in *W*.
- **Pruning the last part of the *SortedCandiListY* (or *SortedListX*).** In Line 10, even though there exists an object in *SortedCandiListY*, we can stop the loop of Lines 10~20 if the total number of objects left in *SortedCandiListY* is not enough to satisfy the frequency-condition (i.e., is less than F/f). Likewise, in Line 6, we stop the hotspot detection process if the total number of objects left in *SortedListX* is less than F/f.

THEOREM 1. *The worst case time complexity of Algorithm Detect2DiHotspots is* $O(N^2)$, *where N is the object frequency of the given region.*

PROOF. In Figure 5, Lines 1~2 take $O(N \cdot \log N)$ time to sort *N* objects. The while-loop starting from Line 6 repeats at most $(N – (N/f) + 1)$ times by using the *pruning the last part of the SortedListX*. Line 7 takes a constant time by using the *swift examination of the MPF condition*. Line 8 takes O(*N*) time by using the *rapid acquisition of the SortedCandiListY*. Lines 10~20 iterate at most $(N – (N/f) + 1)$ times by using the *pruning the last part of the SortedCandiListY*. Line 11 also takes a constant time by using the *swift examination of the MPF condition*. Consequently, the complexity of Algorithm DetectHotspots is $O(N^2)$. □

We can easily extend Algorithm Detect2DiHotspots to the algorithm for three dimensional data. This algorithm, called Algorithm Detect3DiHotspots, is a straightforward extension of Algorithm Detect2DiHotspots. Because of space limitation, we omit the details of Algorithm Detect3DiHotspots in this paper. The following corollary, though the details of the algorithm as well as the proof of it are omitted, is an immediate consequence of the fact that Algorithm Detect3DiHotspots is a straightforward extension of Algorithm Detect2DiHotspots to handle one more dimension in the data.

COROLLARY 1. *The worst case time complexity of Algorithm Detect3DiHotspots is* $O(N^3)$, *where N is the object frequency of the given region.*

For convenience, Algorithm Detect2DiHotspots and Algorithm Detect3DiHotspots will be collectively called Algorithm DetectHotspots if there is no ambiguity.

Note that, depending on the policy to detect hotspots, there can be various algorithms that are different from our Algorithm DetectHotspots. The number of hotspots and their locations found in different algorithms can be different from those found in Algorithm DetectHotspots. However, any hotspot detection algorithm can be used in Algorithm ConstructHotspotTree discussed in the next section as long as if it correctly finds hotspots that satisfy only condition (2) in Definition 2, i.e., any two hotspots found in the current bucket are mutually exclusive.

3.4 Constructing Histograms Based on Hotspots

In this section, we present how to construct our proposed histogram. The proposed method detects hotspots by using Algorithm DetectHotspots, and organizes them into a hierarchy of buckets, i.e., a hotspot tree. We start with bucket B_{root} that corresponds to the minimum bounding rectangle for a given entire data set, and construct a hotspot tree rooted by the node B_{root}. Figure 7 shows the proposed algorithm ConstructHotspotTree that recursively constructs a hotspot tree. This algorithm requires as input a bucket *B*, a set of objects *O*, and the number of buckets N_B to be constructed within *B*. The initial call of the algorithm is ConstructHotspotTree(B_{root}, *DataSet*, *p*), where *DataSet* and *p* are the entire data set and the number of buckets to be constructed in B_{root}, respectively. At each recursive call to ConstructHotspotTree, we first determine the size and minimum object frequency for a hotspot in this recursive step (Lines 1~2), i.e., the input parameters *s* and *f* of Algorithm DetectHotspots. We then detect hotspots in bucket *B* by using Algorithm DetectHotspots (Line 3). The detected hotspots are organized as children of bucket *B* in the hotspot tree (Line 4). Here, for each newly created node *h*, the

```
ALGORITHM ConstructHotspotTree($B$, $O$, $N_B$)
INPUT: $B$ – a bucket that is the root node of a hotspot tree for a data set $O$.
        $O$ – a set of geographic objects in bucket $B$.
        /* We assume that $O$ is either a set of two-dimensional point objects
           or a set of three-dimensional point objects. */
       $N_B$ – the number of buckets to be constructed within $B$.
OUTPUT: a hierarchy of buckets rooted by bucket $B$.
/*  Determine the input parameters $s$ and $f$ of
     Algorithm DetectHotspots. */
1: $s \leftarrow \max(N_B, 2)$;        /*  explained later. */
2: $f \leftarrow \max(N_B, 2)$;        /*  explained later.  */
/*  Detect hotspots within bucket $B$ by using
    Algorithm DetectHotspots.  */
3: SetOfBuckets $\leftarrow$ DetectHotspots(a minimum bounding region
                                             for objects in $B$, $O$, $s$, $f$);
4: Make buckets in SetOfBuckets children of $B$;
/*  $N_B'$ denotes the number of buckets in SetOfBuckets. */
/*  Note that $N_B' \le N_B$. */
5: IF ($N_B' < N_B$) THEN {
6:    FOR EACH bucket $B_i$ in SetOfBuckets DO {
/*  $N_{Bi}$ denotes the number of buckets to be constructed in $B_i$. */
7:       Compute $N_{Bi}$ for bucket $B_i$;      /* explained later. */
8:       IF ($N_{Bi} \ge 1$) THEN {
/*  Detect hotspots within bucket $B_i$ recursively. */
9:          ConstructHotspotTree($B_i$, a data set in $B_i$, $N_{Bi}$);
10:      }
11:   } // END-OF-FOR-EACH
12:}
```

Figure 7: A hotspot tree construction algorithm.

related statistics, i.e., data space and object frequency are maintained within h. We use N_B for the value of f, which makes the number of hotspots detected be less than or equal to N_B according to Property 1. If the number of hotspots detected is equal to N_B, the current recursive step completes its execution and returns. Otherwise, we proceed to construct more buckets that will become the next level nodes in the hotspot tree (Lines 6~11). That is, for each hotspot h found at the current step, we detect hotspots within h. These hotspots will be organized as children of h at the next recursive step. This process continues until no more buckets can be constructed.

Input parameters *s* and *f* of Algorithm DetectHotspots (Lines 1 and 2 in Figure 7). Let us consider $N_B \ge 2$ first. We use N_B for the value of f if $N_B \ge 2$. This is because if we use $f = N_B$, the number of hotspots found in a bucket B, denoted by N_B', does not exceed N_B by Property 1.

Now, consider the value of k that denotes the minimum relative density for a hotspot. In general, k can be any real number greater than or equal to 1. Note that $k = 1$ means the density of a hotspot can be at least the same as the density of its enclosing bucket. It also means $s = f$ by Property 2. This implies that the size of a subregion to be investigated for a hotspot in Algorithm DetectHotspots is $1/N_B$ of the size of B because we use $f = N_B$ as mentioned above. Suppose $k = 1$ and the entire data set has perfectly uniform distribution. Then, our hotspot tree will have only two levels, i.e., all the nodes except the root node are at the same level, which effectively reduces to a histogram similar to the conventional one. Suppose $k = 1$ but the distribution of data is skewed. In bucket B (that is B_{root} in the initial case), consider N_B number of disjoint subregions, each size of which is $1/N_B$ of the size of B. Some of them are dense subregions and the others are sparse subregions. Since sparse subregions are not hotspots by definition, the number of dense subregions will be less than N_B. Then, the recursive step of our algorithm will continue. We will use $k = 1$ in our algorithm in Figure 7, which allows our algorithm to work well regardless of uniform distribution or skewed distribution of data. This implies that N_B is used for the value of s if $N_B \ge 2$. The exceptional case is $N_B = 1$. Since a child bucket with $s = 1$ (i.e., finding a subregion whose size is equal to the size of the enclosing bucket) does not make sense we use $s = 2$ and $f = 2$ in this case.

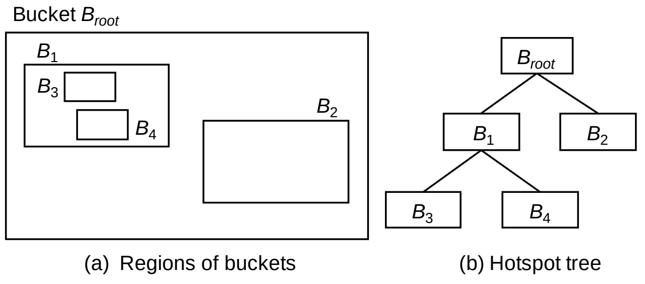


Figure 8: An example of a construction process of a hotspot.

If a value greater than one is used for k, there is a possibility that Algorithm DetectHotspots may not find any hotspot in some recursive step. This case occurs when the data in a bucket within which we want to find hotspots is uniformly distributed. Thus, if we use the value of k greater than one, the algorithm in Figure 7 needs a little modification to handle the case that Algorithm DetectHotspots does not find any hotspot. We will not discuss this case further because we will use $k = 1$, i.e., $s = f = N_B$ if $N_B \ge 2$ in our algorithm.

Compute N_{Bi}, i.e., the number of buckets to be constructed in bucket B_i (Line 7 in Figure 7). If the number of hotspots found in bucket B is less than the predefined number of buckets (i.e., $N_B' < N_B$ in Line 5), then for each bucket B_i constructed at the current step, Algorithm ConstructHotspotTree is invoked recursively to construct more buckets (Line 9). At this time, the number of buckets to be constructed in B_i, denoted by N_{Bi}, needs to be determined (Line 7). We compute N_{Bi} in proportion to the relative skew of B_i defined below.

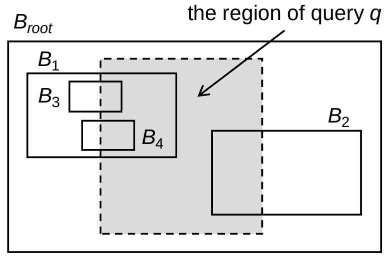
DEFINITION 4 (RELATIVE SKEW OF A BUCKET). *Let a bucket B have n children. For a child node bucket B_i, i = 1, ... , n, of bucket B, the relative skew of B_i, denoted by RelativeSkew(B_i), is defined as follows.*

$$RelativeSkew(B_i) = skew(B_i) \,/ \sum_{i=1...n} skew(B_i).$$

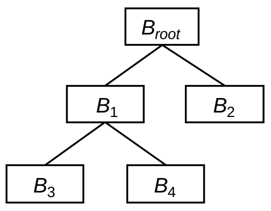
Here, skew(B_i) denotes the skew of distribution in bucket B_i. skew(B_i) is computed as $\sum_r (x_r - \hat{x}_r)^2$ [10], *i.e., the sum of squares of absolute errors for all the locations within B_i. x_r is the real object frequency at location r and $\hat{x}_r$ is the estimate of the object frequency based on the uniform distribution assumption within B_i.*

Note that, in the definition, the unit of location is a cell when the entire data space is considered as a grid, as commonly used in the literature [10, 20].

EXAMPLE 5. *Consider a construction process of a hotspot tree as in Figure 8. Suppose that the number of buckets to be*



(a) Representation in 2-dimensional space



(b) Hotspot tree

Region	Object Frequency	X-range	Y-range	Size
B_{root}	100	0~100	0~60	6000
B_1	40	5~45	25~50	1000
B_2	36	55~95	10~35	1000
B_3	8	15~29	39~48	126
B_4	6	21~35	27~36	126
q	?	25~67	5~55	2100

(c) Information about buckets and query q

Figure 9: Selectivity estimation based on the histogram constructed by our method.

constructed in bucket B_{root} is 4. *Suppose also that within B_{root}, two hotspots are found, i.e., buckets B_1 and B_2 in the figure. Then, we need to construct two more buckets within B_1 and B_2. Each number of buckets to be constructed in B_1 and B_2, i.e., N_{B1} and N_{B2} is computed in proportion to the relative skew of B_1 and B_2. That is,*

$$N_{B1} = 2 \cdot \frac{skew(B_1)}{skew(B_1) + skew(B_2)} \text{ and } N_{B2} = 2 \cdot \frac{skew(B_2)}{skew(B_1) + skew(B_2)}.$$

If the skew of distribution in B_1 is much higher than that in B_2, then N_{B1} = 2 *and N_{B2} =* 0. *Hence, only within B_1, we attempt to find hotspots and organize them as children of B_1, i.e., buckets B_3 and B_4 in the figure.*

Let B_1, B_2, … , B_n be the children of bucket B, and m be the number of buckets to be constructed in some of B_1, B_2, … , B_n. Then, as in Example 5, $N_{Bi} = m \cdot RelativeSkew(B_i)$.

THEOREM 2. *Let $F(N)$ be the worst case computing time for Algorithm DetectHotspots where N is the number of objects in the whole data set. Then the worst case time complexity of Algorithm ConstructHotspotTree is* $O(p \cdot F(N))$, *where p is the number of buckets for a histogram.*

PROOF. Let $T(N)$ denote the computing time of Algorithm ConstructHotspotTree for N objects. If the initial step of ConstructHotspotTree finds h number of hotspots, then $T(N) = T(N_1) + T(N_2) + \ldots + T(N_h) + f(N)$. Here, N_i denotes the number of objects in the i-th hotspot and $f(N)$ denotes the computing time for DetectHotspots. The worst case of the algorithm occurs when only a single hotspot is found, i.e., $h = 1$, whose object frequency is $N - 1$, at every recursive step. In this case, the above equation can be rewritten as $T(N) = T(N - 1) + f(N) = T(N - 2) + f(N - 1) + f(N) = f(N - p + 1) + \ldots + f(N - 1) + f(N)$. Since $F(N)$ is the worst case of $f(N)$, the proof follows. □

Note that the worst case time complexity of Algorithm ConstructHotsptTree that uses Algorithm DetectHotspots described in Section 3.3 is $O(p \cdot N^2)$ for a two-dimensional data set and is $O(p \cdot N^3)$ for a three-dimensional data set. However, if we can come up with a new hotspot detection algorithm whose worst case time complexity is better than Algorithm DetectHotspots in Section 3.3, then the worst case time complexity of Algorithm ConstructHotsptTree can also be improved.

3.5 Estimating Selectivities Using Our Proposed Histogram

In this section, we explain how to compute a selectivity for a given query based on the histogram constructed by our method.

EXAMPLE 6. *Consider a histogram H consisting of five buckets B_{root}, B_1, B_2, B_3, and B_4. Visual representation in two dimensional space and the hotspot tree representation of H are shown in Figures* 9(*a*) *and* 9(*b*), *respectively. The object frequencies, ranges of the* X*-axis, ranges of the* Y*-axis, and sizes of buckets are described in Figure* 9(*c*). *Consider also a query q whose region is* $(25 \le X \le 67,\ 5 \le Y \le 55)$. *Then, the selectivity of q based on H is computed as follows. We first find leaf nodes in the hotspot tree, i.e., buckets B_2, B_3, and B_4 in the figure. Let $size(B_i, q)$ denote the size of B_i that overlaps with the region of query q. Then,*

$$size(B_2, q) = (67 - 55) \cdot (35 - 10) = 300,$$

$$size(B_3, q) = (29 - 25) \cdot (48 - 39) = 36, \textit{ and}$$

$$size(B_4, q) = (35 - 25) \cdot (36 - 27) = 90.$$

Let $est(B_i, q)$ denote the selectivity estimate for the overlapping region between bucket B_i and query q.

$$est(B_i, q) = size(B_i, q)/size(B_i) \cdot freq(B_i).$$

Thus, $est(B_2, q) = 300/1000 \cdot 36 = 10.8$, $est(B_3, q) = 36/126 \cdot 8 = 2.3$, *and* $est(B_4, q) = 90/126 \cdot 6 = 4.3$. *Next, we go up to the parent nodes of buckets B_2, B_3, and B_4, i.e., buckets B_1 and B_{root} in the figure. Now, for each bucket B_1 and B_{root}, check if the selectivities of its all the children have already been computed. Bucket B_1 is the only case. In the region of B_1, selectivity estimates of q for buckets B_3 and B_4 have already been computed. Let B_i' denote the region of B_i that does not overlap with those of its children. Then,* $size(B_1') = size(B_1) - [size(B_3) + size(B_4)] = 748$,

$$size(B_1', q) = size(B_1, q) - [size(B_3, q) + size(B_4, q)] = 374,$$

$$\textit{and } freq(B_1') = freq(B_1) - [freq(B_3) + freq(B_4)] = 26.$$

Since there is no hotspot in B_1', we assume that 26 *objects are uniformly distributed in B_1' whose size is* 748. *So,* $est(B_1', q) = size(B_1', q)/size(B_1') \cdot freq(B_1') = 374/748 \cdot 26 = 13$. *Thus,* $est(B_1, q) = est(B_3, q) + est(B_4, q) + est(B_1', q) = 2.3 + 4.3 + 13 = 19.6$. *Now, since selectivity estimates of all the children of B_{root} have been computed, we can compute the selectivity estimate of B_{root} using the same procedure just mentioned.*

$$size(B_{root}') = size(B_{root}) - [size(B_1) + size(B_2)] = 4000,$$

```
ALGORITHM ComputeSelectivity(H, q)
INPUT: H – a hotspot tree rooted by bucket B_root, q – a query.
OUTPUT: the selectivity of query q.
(1) /* Processing at the leaf nodes: */
  Let SetLeaf be a set of leaf nodes, i.e., buckets, in H
  whose regions overlap with the region of q.
  For each node B_i in SetLeaf, compute
      $est(B_i, q) = size(B_i, q) / size(B_i) \cdot freq(B_i)$.
  If the entire region of q is contained in the region of $\bigcup_i B_i$
      return $\sum_i est(B_i, q)$.
  Otherwise, for each B_i in SetLeaf, go up to the parent node
  and continue Step (2).
(2) /* Processing at the nonleaf nodes: */
  Let SetNonLeaf be a set of nodes, each of which is the parent
  of some node B_i whose est(B_i, q) has been computed.
  If |SetNonLeaf| = 1, return est(B, q), where B is the sole node in SetNonLeaf.
  Otherwise, find a node B that is in the lowest level of
  the hotspot tree among nodes in SetNonLeaf, and compute est(B, q).
  Go up to the parent node, and repeat this step, i.e., Step (2).
```

Figure 10: The selectivity estimation algorithm for our proposed histogram.

$$size(B_{root}', q) = size(B_{root}, q) - [size(B_1, q) + size(B_2, q)] = 1300,$$

$$freq(B_{root}') = freq(B_{root}) - [freq(B_1) + freq(B_2)] = 24, \textit{ and}$$

$$est(B_{root}', q) = 1300/4000 \cdot 24 = 7.8.$$

Thus, $est(B_{root}, q) = est(B_1, q) + est(B_2, q) + est(B_{root}', q)$

$$= 19.6 + 10.8 + 7.8 = 38.2.$$

Let B_c denote a child node of bucket B, and B' denote a region of B except its children. The equation that computes $est(B, q)$, i.e., the selectivity estimate of a query q for bucket B, is as follows.

$$est(B,q) = \begin{cases} \dfrac{size(B,q)}{size(B)} \cdot freq(B) & \textit{if } B \textit{ is a leaf node,} \\ \displaystyle\sum_{c=1\ldots n} est(B_c, q) + \left(\dfrac{size(B',q)}{size(B')} \cdot freq(B') \right) & \textit{otherwise.} \end{cases}$$

Algorithm ComputeSelectivity in Figure 10 computes a selectivity estimate of a given query q using our proposed hotspot tree. Note that Algorithm ComputeSelectivity as well as Algorithm ConstructHotspotTree is not affected by the dimension of data.

4. HOTSPOT FOREST

Until now, our proposed histogram is constructed based on the existence of hotspots only. This strategy may have a small problem when query ranges are in the areas where no hotspot is found. Consider the bucket B_{root} that is the minimum bounding rectangle for the entire data space. Suppose that two hotspots B_1 and B_2 within B_{root} are found by Algorithm DetectHotspots. Let B' be the subregion of B_{root} that does not overlap with B_1 and B_2. According to our algorithms described so far, all the remaining buckets will be constructed within B_1 and B_2, and no bucket will be formed in B'. Note that even though there is no hotspot in B', this does not necessarily mean that data is uniformly distributed in B'. No existence of a hotspot in B' simply means that within B' there is no subregion that satisfies the hotspot conditions in Definition 2. For a query whose range overlaps with B_1 or B_2, our

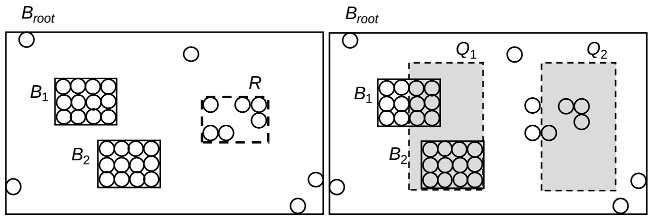


Figure 11: A hotspot tree and two query ranges.

histogram will give a good estimate in general. However, for a query whose entire range or most of whose range overlaps with B', the accuracy of our estimate could depend on the distribution of B'. Figure 11 illustrates this case. Suppose that the minimum object frequency for a hotspot is 7 in Figure 11(a). In the figure, B_1 and B_2 are hotspots while the region R is not. This is because in R there are only 6 objects, which does not satisfy the *frequency-condition* for our hotspot. Consider two query ranges Q_1 and Q_2 shown in Figure 11(b). Our histogram will give a good estimate for Q_1, but will not for Q_2. If a distribution of query ranges (with respect to the locations in the data space) is uniform, i.e. query ranges such as Q_1 and Q_2 will be given equally likely, we also need to care about subregions such as R in the figure.

To alleviate the problem mentioned above, we can use slightly modified approach such that we build more than one hotspot tree, each of which covers a different subregion of the entire data space. Let p be the number of buckets to be constructed for a histogram. Initially, we partition the entire data space into p disjoint segments of approximately equal sizes. Remove segments that have no data. Then, combine two adjacent segments S_i and S_j if $skew(S_i \cup S_j) \leq skew(S_i) + skew(S_j)$ where $S_i \cup S_j$ denotes a segment that results from combining S_i and S_j. This process is repeated until no two adjacent segments are combined. Then, we construct a hotspot tree for each segment resulting from the above process. This set of hotspot trees is called the *hotspot forest*. As in Section 3.4, the number of buckets to be constructed in each hotspot tree is computed in proportion to its relative skew. Note that the worst case time complexity for constructing a hotspot forest does not change, i.e. is equal to the one in Theorem 2. Now, given a range query, a selectivity estimate of the query is the total sum of estimates for each hotspot tree.

The underlying philosophy for building hotspot forest instead of a single hotspot tree can be roughly stated as follows: For a subregion R of size $1/p$ of the entire data space, we want to allocate at least one bucket unless there is any region enclosing R whose skew is lower than the skew of R. Note that when a data set is perfectly uniform, a hotspot forest will be the same as the histogram of the well-known equal-width approach.

Figure 12(a) shows the two-dimensional *Sequoia* data set [25], which is a well-known real-life data set. This data set is publicly available and has been used in performance analysis tasks [6, 30]. Figure 12(b) provides our *hotspot forest* histogram for the *Sequoia* data set. We also present the *MinSkew* [1], *GenHist* [11, 12], and *RK-Hist* [8] histograms for the data set in Figure 12(c), 12(d), and 12(e), respectively.

5. PERFORMANCE EXPERIMENTS

In order to study the effectiveness of our proposed histogram, we have conducted extensive experiments with real-life data as

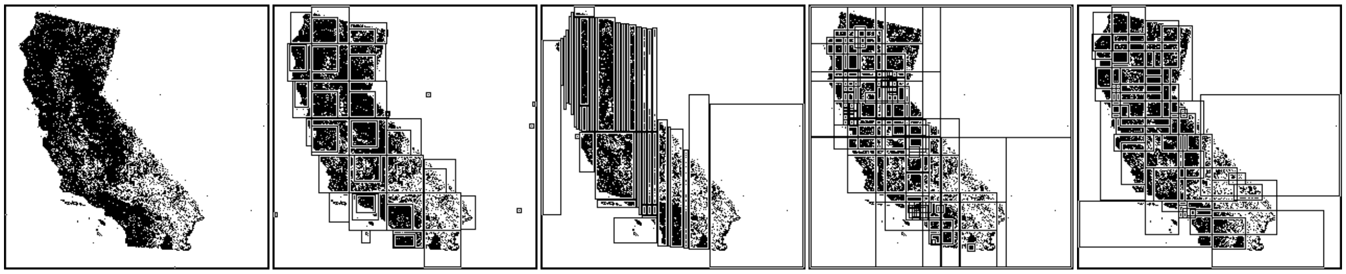

(a) *Sequoia* data (b) Our proposed histogram (c) *MinSkew* histogram (d) *GenHist* histogram (e) *RK-Hist* histogram

Figure 12: *Sequoia* two-dimensional data and various histograms (50 buckets).

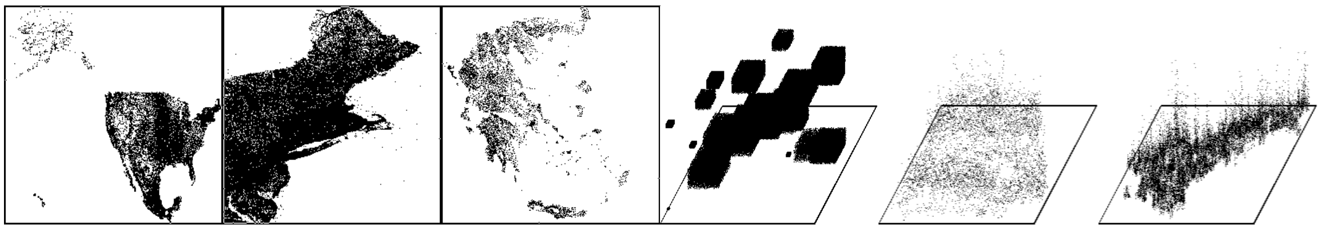

(a) *Digital Chart of the World* data (b) *North East* data (c) *Greece Cities* data (d) *Cluster* (3D) data (e) *Geevor* (3D) data (f) *Sample* (3D) data

Figure 13: Geographic data sets.

well as synthetic data, and compared the accuracy of our results with those of the *MinSkew* [1], *GenHist* [11, 12], and *RK-Hist* [8] histograms. (*MinSkew* and *GenHist* require initially to divide the data space into a grid of cells. We used a grid with 10^4 cells as in [1].) In evaluating our method, we have used *hotspot forests* described in the previous section. All experiments reported in this section have been performed on a Windows server 2003 workstation with two Xeon 3GHz quad-core processors and 8GB memory. The algorithms have been implemented in Visual C++ 2008.

Data sets. We generated synthetic data sets, referred to *Cluster* data sets, with many clusters and therefore high correlations between attributes, as suggested in [12]. The parameters of the data generator are i) the dimension of the data space, ii) the total number of clusters, and iii) the maximum size of a cluster, represented by the ratio of the cluster size to the domain size, set to 2 (and 3), 50, and 1%, respectively in our experiments. The clusters each defined as a hyper-rectangle are randomly located within the data space and data within each cluster are randomly distributed. Each synthetic data set contains 1,000,000 data objects in the space of $[1, 1000]^d$, where d is the dimension of the data. For our real-life data experiments, we used the following data sets: i) the *Sequoia* data set [25], which contains 62,556 locations in California. ii) the *Digital Chart of the World* data set[1], which contains 19,499 populated places in the United States of America and Mexico. iii) the *North East* data set[2], which contains 123,593 postal addresses in three metropolitan areas, i.e., New York, Philadelphia, and Boston. iv) the *Greece Cities* data set[3], which contains 5,922 cities and villages in Greece. v) the *Geevor* data set from the Practical Geostatistics 2000 [7], which contains 5,445 locations of tin mines in Cornwall, England and the grade of tin from each tin mine. vi) the *Sample* data set from the Practical Geostatistics 2000 [7], which contains 21,577 locations of gold mines and the degree of gold from each gold mine. Figure 13 shows all the data sets except that shown in Figure 12(a). Note that the *Sequoia*, the *Digital Chart of the World*, the *North East*, and the *Greece Cities* data sets are two dimensional data sets while the *Geevor* and the *Sample* data sets are three dimensional data sets. The *Cluster* (2D) and the *Cluster* (3D) data sets are synthetic data sets, each of which is two dimensional and three dimensional, respectively.

Buckets, Quality Measure, and Test Queries. To test the accuracy of histograms with various number of buckets, we constructed histograms with various number of buckets (50~1000 buckets). Our comparisons are based on the average relative error, which is commonly used as a performance metric in the selectivity estimation, described below: Let θ be the actual object frequency of a query q, and θ' be the estimated object frequency of q by a histogram. Then, the absolute error e^{abs} and the relative error e^{rel} are defined as follows:

$$e^{abs} = |\theta - \theta'|. \quad e^{rel} = e^{abs} / \max\{1, \theta\} = |\theta - \theta'| / \max\{1, \theta\}.$$

For a set of queries, the average relative error E_{rel} is defined as the sum of the relative errors for all the queries divided by the number of queries. For the test queries, we used 100,000 queries in each experiment, whose regions are randomly located within the data space.

In the experiments, the performance of the histograms was evaluated using the following parameters:

(1) The number of buckets in the histogram.

(2) The size of a query region (simply called the query size).

Figure 14 shows how varying the number of buckets affects the performance of histograms for the real-life data sets and the synthetic data sets. Note that the Y-axis is shown on a log scale and the size of queries is set to 5%. In general, average relative errors tend to be reduced in all the methods with the increasing number of buckets. This is because when the number of buckets increases, more accurate statistics can be obtained. Note that the histograms of the proposed method denoted by *STHistogram* work significantly better than those of other methods in many cases. The primary reason for the noticeable improvement is that

[1, 2, 3]http://www.rtreeportal.org

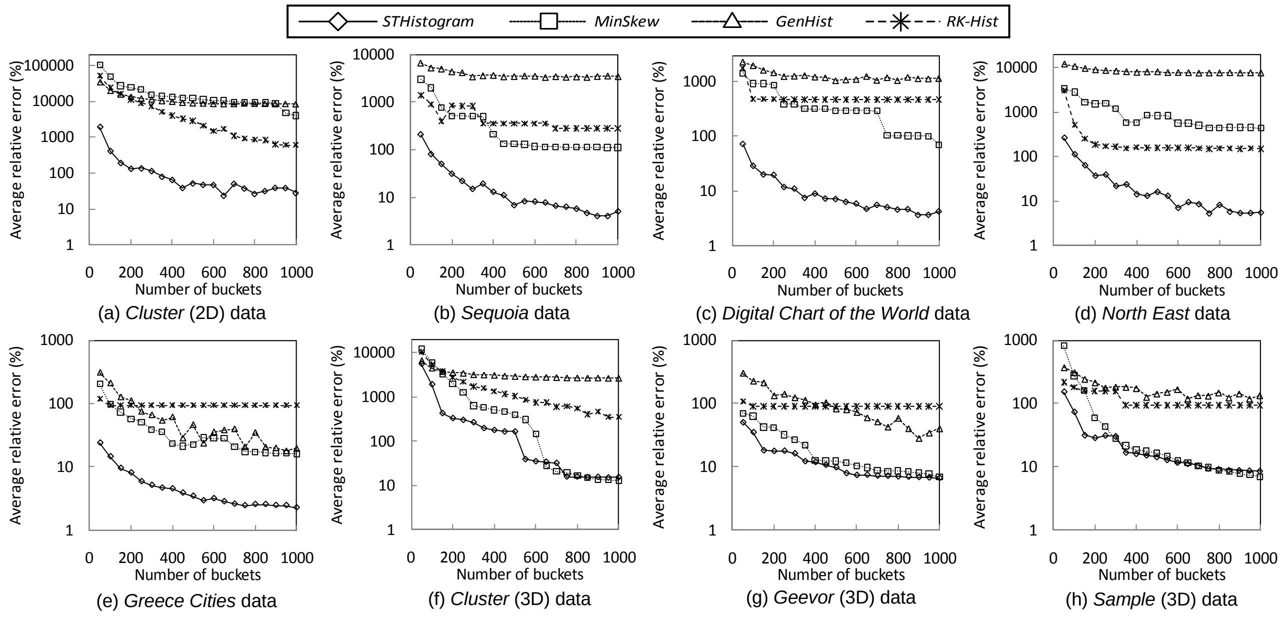


Figure 14: Average relative errors for varying number of buckets.

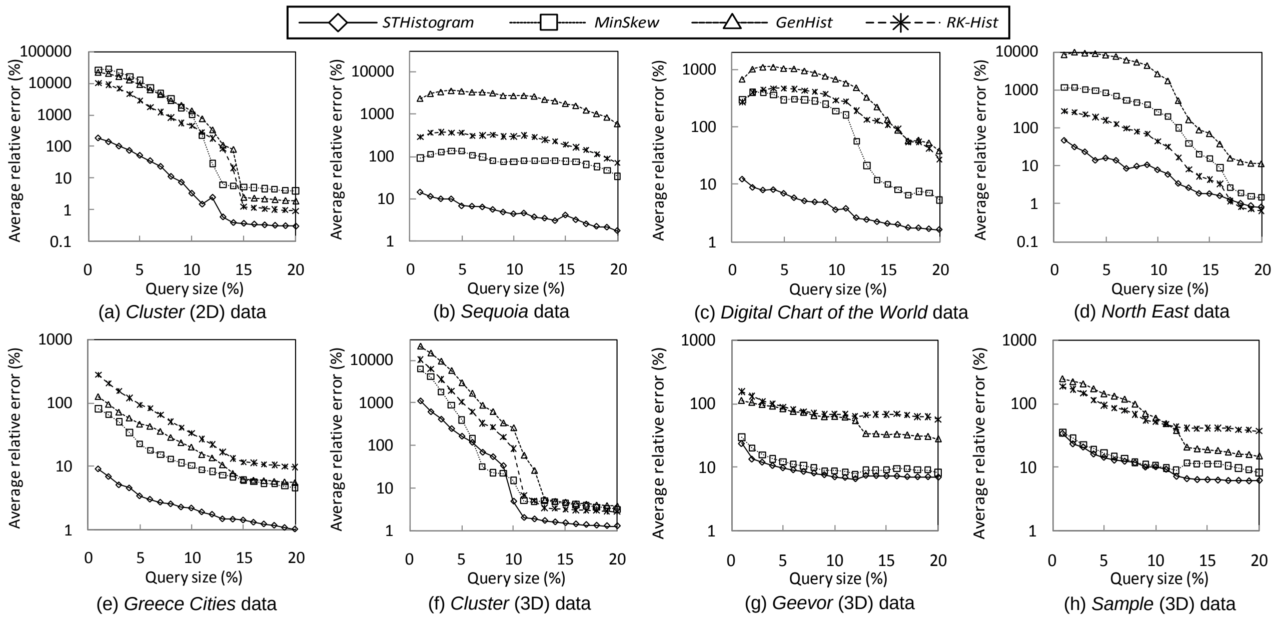


Figure 15: Average relative errors for varying query sizes.

our method effectively handles the clusters that may significantly degrade the accuracy of the histograms. That is, our method effectively detects hotspots and exploits them in constructing buckets. Figure 15 shows how varying the query sizes affect the performance of the histograms for the various data sets. Note that the Y-axis is shown on a log scale and the number of buckets is set to 500. In this figure, as the query size increases, the accuracy of the histogram tends to increase in all the methods. This is because when the query size increases, the number of buckets that are fully contained in the query region also increases in the methods. That is, the effect of buckets that partially overlap with the query region, which are the sources of incorrect selectivity estimation, reduces with the increase of the query size. As seen in the figures in Figure 15, the performance of the proposed STHistogram is much better than other methods in many cases. *MinSkew* shows relatively good accuracy for three-dimensional data sets, as shown in Figures 14(f)-(h) and 15(f)-(h). However, as noted in [1], the accuracy of *MinSkew* is affected by the number of grid cells. Sometimes, the accuracy of *MinSkew* considerably decreases, when the number of grid cells increases. In the experiments with the 10^6 number of grid cells, we have found that the average relative errors of *MinSkew* significantly increase, compared with those in Figures 14(f)-(h) and 15(f)-(h).

Overall, our experimental results show that the proposed STHistogram provides better accuracy than other existing methods for real-life data sets as well as synthetic data sets.

6. CONCLUSIONS

The histogram, which is a simple representation for distribution of a large data set, is widely used for selectivity estimation that has many applications for various types of query processing. Estimates for the histogram buckets that partially overlap with the query region are computed based on the assumption that all the objects in a bucket are uniformly distributed. However, it has been shown to be intractable to organize histogram buckets such that data objects in every bucket are uniformly distributed. Thus, in most heuristic histogram methods, there often exist clusters of data objects in the histogram buckets, which degrades the accuracy of the estimates. In this work, we have attempted to organize histogram buckets in such a way that the degree of skew due to clusters of data objects can be minimized. Our focus is on the histograms for two or three dimensional geographic data objects that have many applications in practice, such as geographic information processing. Through extensive performance experiments, we have shown that our proposed method provides better performance than other existing methods. Our proposed method does not require any user-provided parameters to construct a histogram, which is not the case in some existing methods. In our proposed method, the parameters *s* and *f* used to specify the size and the minimum object frequency of the hotspot are dynamically determined for each bucket when a histogram is constructed.

7. ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2009-0083055). We also thank Dr. Kyoung-Gu Woo and Data Analytics Project members (SAIT, Samsung Electronics) for their support.

8. REFERENCES

[1] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *SIGMOD*, pages 13-24, 1999.

[2] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: a multidimensional workload-aware histogram. In *SIGMOD*, pages 211-222, 2001.

[3] N. Bruno, S. Chaudhuri, and L. Gravano. Top-*k* selection queries over relational databases: mapping strategies and performance evaluation. *ACM Trans. Database Syst.*, 27(2):153-187, 2002.

[4] S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE*, page 64, 2006.

[5] S. Chaudhuri and L. Gravano. Evaluating top-*k* selection queries. In *VLDB*, pages 397-410, 1999.

[6] Y. J. Choi and C. W. Chung. Selectivity estimation for spatio-temporal queries to moving objects. In *SIGMOD*, pages 440-451, 2002.

[7] I. Clark and W. Harper. Practical Geostatistics 2000. Ecosse North America Llc, Columbus, Ohio, USA, 2000.

[8] T. Eavis and A. Lopez. RK-Hist: an r-tree based histogram for multi-dimensional selectivity estimation. In *CIKM*, pages 475-484, 2007.

[9] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *ACM Trans. Database Syst.*, 27(3):261-298, 2002.

[10] S. Guha, K. Shim, and J. Woo. REHIST: relative error histogram construction algorithms. In *VLDB*, pages 300-311, 2004.

[11] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *SIGMOD*, pages 463-474, 2000.

[12] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *VLDB J.*, 14(2):137-154, 2005.

[13] P. J. Haas and A. N. Swami. Sequential sampling procedures for query size estimation. In *SIGMOD*, pages 341-350, 1992.

[14] Y. E. Ioannidis. The history of histograms. In *VLDB*, pages 19-30, 2003.

[15] Y. E. Ioannidis. Query Optimization. *ACM Comput. Surv.*, 28(1):121-123, 1996.

[16] I. Kamel and C. Faloutsos. On Packing R-trees. In *CIKM*, pages 490-499, 1993.

[17] J. Lee, D. Kim, and C. Chung. Multi-dimensional selectivity estimation using compressed histogram information. In *SIGMOD*, pages 205-214, 1999

[18] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, pages 448-459, 1998.

[19] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multidimensional queries. In *SIGMOD*, pages 28-36, 1988.

[20] S. Muthukrishnan, V. Poosala, and T. Suel. On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. In *Int'l Conf. Database Theory*, pages 236-256, 1999.

[21] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41-82, 2005.

[22] V. Poosala and Y. E. Ioannidis. Estimation of query-result distribution and its application in parallel-join load balancing. In *VLDB*, pages 448-459, 1996.

[23] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, pages 486-495, 1997.

[24] U. Srivastava, P. J. Haas, V. Markl, N. Megiddo, M. Kutsch, and T. M. Tran. ISOMER: consistent histogram construction using query feedback. In *ICDE*, page 39, 2006.

[25] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The SEQUOIA 2000 storage benchmark. In *SIGMOD*, pages 2-11, 1993.

[26] J. Sun, Y. Tao, D. Papadias, and G. Kollios. Spatio-temporal join selectivity. *Inf. Syst.*, 31(8):793-813, 2006.

[27] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. In *ICDE*, pages 417-428, 2003.

[28] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *SIGMOD*, pages 428-439, 2002.

[29] J. S. Vitter, M. Wang, and B. R. Iyer. Data cube approximation and histograms via wavelets. In *CIKM*, pages 96-104, 1998.

[30] X. Zhou, D. J. Abel, and D. Truffet. Data partitioning for parallel spatial join processing. *GeoInformatica*, 2(2):175-204, 1998.