# Density Estimation for Spatial Data Streams

Cecilia M. Procopiuc and Octavian Procopiuc

AT&T Shannon Labs, Florham Park, NJ 07950, USA
magda@research.att.com
oprocopiuc@gmail.com

**Abstract.** In this paper we study the problem of estimating several types of spatial queries in a streaming environment. We propose a new approach, which we call Local Kernels, for computing density estimators by using local rather than global statistics on the data. The approach is easy to extend to an on-line setting, by maintaining a small random sample with a kd-tree-like structure on top of it. Our structure dynamically adapts to changes in the locality of data and has small update time. Experimental results show that the proposed algorithm returns good approximate results for a variety of data and query distributions. We also show that it is useful in off-line computations, as well.

## 1 Introduction

We consider the problem of estimating the data distribution and query selectivity for spatial data streams. More precisely, we assume the so-called *cash register* model in which data points are inserted into the set, but they are never deleted. Each data point is a multidimensional real-valued tuple. In this paper, we propose new methods for maintaining an approximate density function on the data, by computing kernel density estimators in an on-line setting. We then use this information in order to approximate the selectivity of range queries.

As in all data stream applications, algorithms for computing the desired statistics must satisfy certain conditions. First, they must require only one pass over the data, and use only a small amount of space compared to the size of the dataset. In addition, processing an update should be fast, as in many applications new tuples arrive with high frequency. And finally, answering queries should be both fast and accurate.

The data stream model has become popular in recent years, motivated by applications that deal with massive information such as Internet and phone traffic log analysis, financial tickers, ATM and credit card operations, sensor networks, etc. Although in many such applications the data is stored and archived, its volume makes it prohibitively expensive to access. Thus, it is often necessary to monitor the contents of very large databases in an incremental, on-line fashion. In addition, for many application domains the ability to answer queries as the data arrives is crucial for mission-critical tasks such as fraud detection or financial transactions.

There is already a large body of literature that studies approximation algorithms for computing various statistics and aggregate queries over data streams. For example, [12,17] propose methods for computing approximate quantiles, and [4,3] estimate on-line self-join and multi-way join sizes. A recent result by Das et al. [9] addresses

C. Bauzer Medeiros et al. (Eds.): SSTD 2005, LNCS 3633, pp. 109–126, 2005.

the problem of computing on-line spatial joins and range queries. Summarization techniques such as sketches [7,21], wavelets [11,13], and histograms [10,14,22] have been proposed as a means of answering more complex queries, such as range queries. However, all these methods make the assumption that each tuple in the data stream has attribute values from some finite universe $\{1, \ldots, U\}$ (for simplicity, we henceforth refer to them as discrete methods). While this is a reasonable restriction for many types of data, it is not always feasible to assume such a priori knowledge of it. For example, in the case of spatial, temporal or multimedia datasets, objects are represented as feature vectors with real-valued attributes. Monitoring such data via discrete on-line methods requires an initial discretization of the objects. The accuracy of the result then depends not only on the guarantee provided by the algorithm, but also on the discretization grid. The higher the accuracy desired, the larger the size $U$ of the discretized universe, and the slower the method (since its space and update time almost always depend on $U$). Moreover, the problem worsens as the dimensionality of the data increases.

In the experimental section we will discuss an application in which the goal is to maintain statistical information of network measurements over time. The data consists of real-valued tuples representing the current state of AT&T's backbone network, reflecting delay times between pairs of servers. Each tuple is an aggregate of measurements taken during a fifteen-minute interval, with multiple measurements generated each minute. Because of the size of the data, only a small amount of it is stored on disk, with older data being moved to tape, which makes accessing it a difficult task. However, it is often the case that only an approximate view of it will suffice, usually to be used for comparison purposes. Thus, our proposed algorithm offers a way to generate a low-storage data summary that is nonetheless powerful enough to answer queries with relatively small error.

Computing summary statistics for real-valued datasets has also been extensively studied in the literature. The Min-Skew [1], MHIST [19] and GENHIST [15] algorithms are histogram-based methods that estimate the selectivity of multi-dimensional range queries. In addition, kernel density algorithms have been proposed in [5,15]. However, all these methods either require multiple passes over the data, or a large enough memory (in relation to the overall data size). The multi-dimensional kernel estimator proposed in [15] is a one-pass algorithm that comes closest to being suitable for an on-line environment. However, its accuracy crucially depends on the computation of approximate values for the standard deviation along each dimension. Such computation requires storing a significant sample of the data. In addition, the proposed algorithm assumes that all queries are asked after the entire data has been seen once.

*Our Contributions.* In this paper we propose a new method for maintaining approximate data distributions on real-valued multi-dimensional data streams. We then use this information in order to estimate the selectivity of range queries arriving in an on-line fashion. To the best of our knowledge, this is the first algorithm that estimates range selectivity over real-valued data streams.

Our approach is to compute kernel density functions and maintain them over insertions and deletions from the sample set of kernel centers (thus making it suitable for on-line computations). We propose new methods for computing the *kernel bandwidths*, by estimating the standard deviations for each kernel only for points that fall in a local-

ity of the kernel center. More precisely, we will maintain a kd-tree-like structure and we will compute separate density functions in each leaf of the tree. We then approximate the standard deviations only for the data points in the corresponding cell. As the kd-tree structure changes, we use previous density information from nearby cells, as well as newly arriving data in order to maintain the standard deviations in the new leaves.

This allows us to achieve good query accuracy while still using only a very limited amount of memory. Previous methods used global standard deviations to compute kernel bandwidths. As we discuss in the experimental section, such approaches usually require larger samples to achieve good accuracy; they may also suffer from an oversmoothing effect of the density estimators. Gunopulos et al. [15] suggested replacing global statistics by local ones, by first clustering the data. However, no experimental data was provided. Moreover, such an approach would be more difficult to adapt to an on-line setting, than the one we propose in this paper. We note that our Local Kernels algorithm is of independent interest and could also be used for off-line applications in order to compute locality-sensitive statistics. We also compare our method with a state-of-the-art on-line algorithm for discrete data, and conclude that it is significantly faster and generally more accurate. Hence, the Local Kernels method is a competitive alternative to current approaches for summarizing discrete on-line data.

Furthermore, the capability to compute on-line approximate density functions can lead to algorithms for other problems, such as maintaining an approximate visualization of the dataset, or detecting high-density areas. The latter is a generalization of the notion of heavy hitters, which were defined for discrete one-dimensional datasets. In general, any off-line application that uses the underlying distribution of the data in order to compute some fast summary statistics can also be translated into a streaming application using our proposed on-line density estimators.

## 2 Preliminaries

*Data and Query Model.* We assume that the dataset is a stream of tuples $\langle p_1, p_2, \ldots \rangle$, where each $p_i \in \mathbb{R}^d$ is a $d$-dimensional point. The points are indexed in the order in which they arrive. We focus on the so-called *cash register* model for data streams, in which points are inserted, but not deleted from the database. This is the natural model for the two real-life applications we consider, in which the data consists of network and weather measurements accumulated over certain periods of time. We will briefly discuss how our approach can be extended to the *turnstile* model, which allows points to be both inserted and deleted from the stream.

We focus on range selectivity queries, defined as pairs of type $Q_i = \langle i, R_i \rangle$, where $R_i$ is a $d$-dimensional hyper-rectangle (see the discussion in Section 4 for extensions to other queries). The selectivity of $Q_i$ is defined as $sel(Q_i) = |\{p_j | j \leq i,\ p_j \in R_i\}|$. In other words, it is the number of points that have arrived up to time step $i$ and that lie inside the query range $R_i$. We study the problem of estimating $sel(Q_i)$, under the assumption that queries arrive in a continuous stream, which is interleaved with the data stream. A query $Q_i$ must be answered before the point $p_{i+1}$ is processed. This requirement arises from the fact that processing the point $p_{i+1}$ changes a subset of the statistics we maintain, and thus has the potential to affect the outcome of $Q_i$. In

practice, if the frequency with which points arrive is too high relative to query time, we can tolerate the processing of a small number of subsequent points, as the changes they induce are not significant. Moreover, as we explain below, processing a point usually affects only a constant number of statistical values, and thus can be handled 'out of sync' by using a small amount of extra memory.

One significant property of our algorithm is that it does not require a priori knowledge of the overall size of the data stream, nor of the range of values along each dimension. The latter is important in any approach that tries to discretize the data first, while the former is often required by discrete on-line methods.

*Kernel Density Estimators.* The problem of estimating an underlying data distribution is a central theme in statistics research [8,20]. Kernel estimators are statistical techniques for approximating the probability distribution, by generalizing random sampling. The first step is to compute a uniform random sample of the data, and to assign each sample point (also called *kernel center*) a weight of one. The second and crucial step is to distribute the weight of each point in the space around it according to a *kernel function*. In general, kernel functions distribute most of the weight over the area in the vicinity of the center, and taper off smoothly to zero as the distance from the center increases. However, in practice it is easier to use non-smooth kernel functions that are zero outside a given area. The study in [8] shows that the shape of the kernel function does not significantly affect the quality of the approximation.

In this paper we will use the Epanechnikov kernel function, which was also employed in [15]. More precisely, let $S = \{s_1, \ldots, s_m\}$ be a random subset of the data. Then the underlying probability distribution is approximated by the function

$$f(x) = \frac{1}{m} \sum_{i=1}^{m} k(x - s_i),$$

where $x = (x_1, \ldots, x_d)$ and $s_i = (s_{i1}, \ldots, s_{id})$ are $d$-dimensional points, and

$$k(x_1, \ldots, x_d) = 0.75^d \frac{1}{B_1 B_2 \ldots B_d} \prod_{j=1}^{d} \left(1 - \left(\frac{x_j}{B_j}\right)^2\right)$$

if $|\frac{x_j}{B_j} < 1|$ for all $j$, and 0 otherwise. See Figure 1 (a).

The parameters $B_1, \ldots, B_d$ are referred to as the *kernel bandwidth* along each dimension. Choosing the right values for these parameters is the crucial step in computing kernel estimators, as they determine the accuracy of selectivity computations. No efficient solution exists for finding optimal bandwidths. The problem has been addressed in [20], which proposes using the following rule: $B_j = \sqrt{5}\sigma_j m^{-1/(d+4)}$, where $\sigma_j$ is the standard deviation of the sample along the $j$th dimension. Note that this rule implies that the same $d$ parameter values are used for all kernel functions, in other words the local distribution around each center is assumed to be identical. Moreover, the accuracy of the method depends on how closely $\sigma_j$ approximates the standard deviation of the entire data. Good approximations require large values for $m$ (the sample size).

In this paper we will use a slightly different approach. We build a kd-tree-like structure on top of the sample $S$, and assume that each sample point $s_i$ is the centroid of the

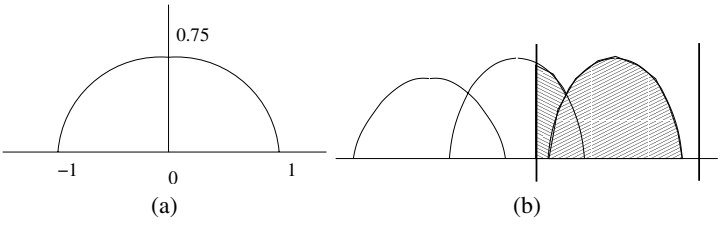


**Fig. 1.** One-dimensional kernels: (a) Kernel function, $B = 1$; (b) Contribution of multiple kernels to estimate of range query

dataset in its leaf. We then maintain a set of $d$ values $\sigma_{ij}$, $1 \leq j \leq d$, that are generally distinct for each sample point, such that $\sigma_{ij}$ approximates the standard distribution of the points in the cell of $s_i$, along dimension $j$. Note that we can update these values in an on-line fashion, by looking at all points in the cell (rather than just a sample). The detailed description of the procedure is given in the next section.

*Computing range selectivities.* Let $R = [a_1, b_1] \times \ldots \times [a_d, b_d]$ be a range selectivity query. Let $T_i$ denote the subset of points in the tree leaf associated with sample point $s_i = (s_{i1}, \ldots, s_{id})$, and let $B_{i1}, \ldots, B_{id}$ be the kernel bandwidths for that leaf. Then the selectivity of $R$ with respect to $T_i$ is approximated in time $O(d)$ as follows:

$$sel(R, T_i) \approx |T_i|(0.75)^d \frac{1}{B_{i1} \ldots B_{id}} \times \prod_{j=1}^{d} \int_{[a_j, b_j]} \left(1 - \left(\frac{x_j - s_{ij}}{B_{ij}}\right)^2\right) dx_d \ldots dx_1.$$

The overall selectivity is then computed in time $O(dm)$ as $sel(R) \approx \sum_{i=1}^{m} sel(R, T_i)$. See Figure 1 (b) for an illustration of the contribution of three kernel functions to the selectivity mass of a range query (in the one-dimensional case).

## 3 On-line Algorithm

In this section we describe our algorithm for maintaining a set of kernel density estimators that closely approximates the underlying data distribution of a spatial data stream. An important contribution of our approach is to design kernel estimators that use local statistics on the data in order to define the weight distribution functions. The advantage of using such estimators is that local statistics can be maintained very accurately with only a small amount of memory, and in an on-line fashion. By contrast, previous methods approximated global standard deviations and required a large sample size in order to ensure reasonable accuracy. In addition, our experimental results indicate that using local statistics improves the quality of the results. We will call our approach Local Kernels.

We propose using a kd-tree-like structure to partition the kernel centers, and define the neighborhood of a center to be its corresponding leaf. The algorithm will maintain the (approximate) standard deviations for each subset of data points in a leaf. Other partitioning schemes could also be employed to define such neighborhoods. However,

as we show at the end of Section 3.1, the tree structure we consider exhibits certain properties that make it particularly suitable for our task.

Let $S$ be a uniform sample of the data seen so far. The kernel centers are defined to be the points of $S$. We maintain a hierarchical decomposition tree on $S$, denoted $\mathcal{T}(\mathcal{S})$, with the following properties: each leaf of $\mathcal{T}(\mathcal{S})$ corresponds to a (possibly unbounded) axis-parallel hyper-rectangle in $\mathbb{R}^d$, so that there is exactly one point of $S$ in each leaf; any two leaves are disjoint; and the union of all leaves is $\mathbb{R}^d$. For the initial random sample $S$, $\mathcal{T}(\mathcal{S})$ is a kd-tree. As points are inserted and deleted from $S$, the structure $\mathcal{T}(\mathcal{S})$ will no longer be a kd-tree, but it will maintain the above properties. For each sample point $s_i \in S$ we also maintain $d+1$ values: $\tau_i$, which is the (approximate) number of stream points that lie in the leaf corresponding to $s_i$, and $\sigma_{ij}, 1 \leq j \leq d$, which approximate the standard deviations of the points in the leaf of $s_i$ along each dimension.

### 3.1 Random Sampling

The random sample $S$ is chosen using the reservoir sampling method of [24]. More precisely, point $p_i$ is chosen in $S$ with probability $|S|/i$ (recall that the points are indexed in the order in which they arrive in the stream). If $p_i$ is chosen, a random point of $S$ is deleted. Vitter [24] proves that using this technique guarantees that $S$ is always a uniform random sample of the data seen so far.

A powerful probabilistic result by Vapnik and Chervonenkis [23] allows us to compute the size of the random sample $S$ for which certain estimation errors are guaranteed with high probability. More precisely, we can prove the following.

**Theorem 1.** *Let $T$ be the data stream seen so far, and let $S \subseteq T$ be a random sample chosen via the reservoir sampling techqnique, such that $|S| = \Theta(\frac{d}{\epsilon^2}\log\frac{1}{\epsilon} + \log\frac{1}{\delta})$, where $0 < \epsilon, \delta < 1$. Then with probability $1-\delta$, for* any *axis-parallel hyper-rectangle $Q$ the following is true:*

$$|sel(Q) - sel(Q,S)\frac{|T|}{|S|}| \leq \epsilon|T| \tag{1}$$

*where $sel(Q) = |Q \cap T|$ is the selectivity of $Q$ with respect to the data stream seen so far, and $sel(Q,S) = |Q \cap S|$ is the selectivity of $Q$ with respect to the random sample.*

*Proof.* Consider the set system $(T, \mathcal{H}(\mathcal{T}))$, where each $H \in \mathcal{H}(\mathcal{T})$ is a subset of $T$ lying in an axis-parallel hyper-rectangle (for an introduction to the theory of set systems we refer the reader to, e.g., the book [18]). It is well known that the VC-dimension of this system is $2d$. By the result of [24], $S$ is a uniform random sample of $T$. Then the main theorem of [23] states that, if $S$ has the size specified above, with probability $1-\delta$ equation 1 is true for any axis-parallel hyper-rectangle $Q$.

Theorem 1 implies that with high probability we can estimate the selectivity of any range query via the simple random sampling method, and achieve an *additive error*

$\epsilon|T|$. In the following, we will be interested in selectivity estimators with small *relative errors*, where the relative error for a query $Q$ is defined as

$$\frac{|sel(Q) - estimated\_sel(Q)|}{\max\{sel(Q), 1\}}.$$

Hence, for the random sampling estimator, defined as

$$estimated\_sel(Q) = sel(Q, S)\frac{|T|}{|S|},$$

the relative error is $\epsilon\frac{|T|}{\max\{sel(Q),1\}}$. Note that this error is small for queries with high selectivity, but it can grow as large as $\Theta(|T|)$ if $sel(Q) = O(1)$. This behavior of the random sampling estimator is well known in practical applications, and has justified the study of more sophisticated estimators. As mentioned before, kernel density estimators can be viewed as a generalization of random sampling, in which the points in $S$ distribute their weight over a local neighborhood, and the selectivity is estimated as an integral of the weight distribution functions. This smoothing technique improves the accuracy of the approximation, especially for ranges of small selectivity.

An immediate consequence of Theorem 1 is that, with high probability, no leaf of the decomposition tree $\mathcal{T}(\mathcal{S})$ contains more than $2\epsilon|T|$ data stream points. Indeed, let $L$ be the axis-parallel hyper-rectangle associated with a leaf of $\mathcal{T}(\mathcal{S})$. By construction of the tree, $sel(L, S) = 1$. Then if we assume that $sel(L) \geq 2\epsilon|T|$, equation 1 implies $|S| \leq 1/\epsilon$, a contradiction. Hence, $\mathcal{T}(\mathcal{S})$ induces a good partitioning of the data stream, in the sense that no leaf is too dense. This is particularly important if the underlying data is clustered in a reasonably large number of dense subsets, as it ensures that points from different clusters fall in different leaves. This in turn means that our estimates $\sigma_{ij}$ of the standard deviations for points in each leaf are close to the real values for that region.

The above observations justify our choice of using a kd-tree-like structure for partitioning the data stream. Note that other means of defining local neighborhoods for the sample points can also be employed. For example, one could use the Voronoi diagram of $S$, and compute the values $\tau_i$ and $\sigma_{ij}$ for each subset of points lying in a Voronoi cell. However, in order to ensure that no cell of the partition is too dense (in the sense discussed above), we have to significantly increase the sample $S$. More precisely, $|S|$ must have a linear dependence on the VC-dimension of the corresponding set-system; this is asymptotically larger for any other reasonable decomposition schemes.

In addition to this well-balanced property in terms of data density in each cell, $\mathcal{T}(\mathcal{S})$ also has the advantage of being easy to maintain. As we show below, we can update $\mathcal{T}(\mathcal{S})$ in time $O(|S|)$ under insertions and deletions from $S$.

### 3.2 Updating $\mathcal{T}(\mathcal{S})$

For ease of presentation, we introduce the following notations. Let $box(v)$ be the axis-parallel hyper-rectangle associated with a node $v$ of $\mathcal{T}(\mathcal{S})$. If $v$ is an internal node, let $h(v)$ denote the hyper-plane orthogonal to a coordinate axis that divides $box(v)$ into the

two smaller boxes associated with the children of $v$. Let $leaf(s_i)$ be the leaf containing sample point $s_i$. We maintain $\tau_i \approx$ number of stream points contained in $leaf(s_i)$ and

$$\Sigma_{ij} \approx \sum_{p \in leaf(s_i)} (p_j - s_{ij})^2,\ 1 \leq j \leq d.$$

Then $\sigma_{ij} = \sqrt{\Sigma_{ij}/(\tau_i - 1)}$ is the approximate standard deviation of the points in $leaf(s_i)$ along dimension $j$ (assuming $s_i$ is the centroid of the distribution).

Let $p$ be the current point in the data stream. If $p$ is not selected in the sample $S$, we find the leaf that contains $p$ - say this is $leaf(s_i)$. Then we increment $\tau_i$ by one, and we add $(p_j - s_{ij})^2$ to $\Sigma_{ij},\ 1 \leq j \leq d$.

We now consider the case when $p$ is selected in the random sample $S$. Let $q$ denote the point that gets deleted from $S$. The updating procedure first deletes $leaf(q)$ from the tree, and then adds a new leaf corresponding to $p$. We detail each step below.

*Deleting a leaf.* Let $u$ denote the parent node of $leaf(q)$, and let $v$ be the sibling of $leaf(q)$. Without loss of generality, assume that $leaf(q)$ lies to the left of $h(u)$ and $v$ lies to the right of $h(u)$; see Figure 2. The structure of $\mathcal{T}(\mathcal{S})$ will be modified as illustrated in Figure 3: make $v$ a descendant of the parent node of $u$; delete nodes $u$ and $leaf(q)$. Let $\mathcal{N}(q)$ denote the leaves in the subtree of $v$ that have one boundary contained in $h(u)$; we will call these the *neighbors* of $leaf(q)$. The deletion procedure can be viewed as extending the bounding box of each neighbor of $leaf(q)$ past the hyper-plane $h(u)$, until it hits the left boundary of $leaf(q)$. The points that were previously contained in $leaf(q)$ must thus be redistributed among the leaves in $\mathcal{N}(q)$, and the corresponding $\tau$ and $\Sigma$ values must be updated for all these leaves. The procedure

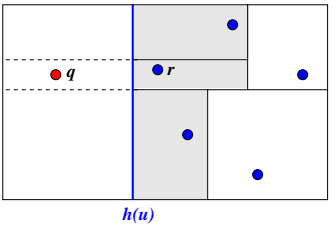


**Fig. 2.** Deleting $leaf(q)$ means extending the bounding boxes of leaves in $\mathcal{N}(q)$ (represented as gray rectangles)

for updating the $\tau$ values is simple: for each leaf in $\mathcal{N}(q)$, we increment its $\tau$ value by the selectivity of its (expanded) bounding box with respect to the points contained in $leaf(q)$. As long as the kernel function for $leaf(q)$ is a good model for the distribution of points inside the leaf, $\tau$ remains a good approximation for the actual number of points inside each leaf of $\mathcal{N}(q)$. However, updating the $\Sigma$ values requires more information than we store. This is because points in *leaf*$(q)$ will contribute differently to the standard deviations of the (expanded) leaves in $\mathcal{N}(q)$, based on their relative position to the centroids of those leaves. Let $r$ be a point so that *leaf*$(r) \in \mathcal{N}(q)$, and for a fixed

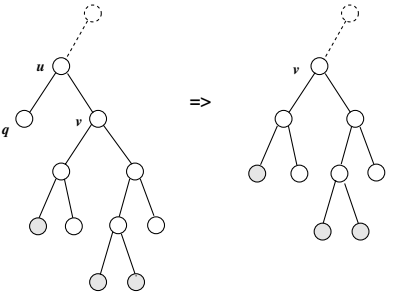


**Fig. 3.** Deleting $leaf(q)$ from the tree. Gray nodes correspond to $\mathcal{N}(q)$

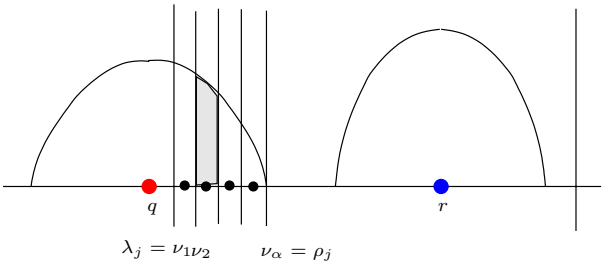


**Fig. 4.** Updating $\Sigma_{rj}$ by discretizing the intersection of $box_e(r)$ and the kernel of $q$ along dimension $j$ (the gray area represents $wt_2$)

dimension $j$, let $[\lambda_j, \rho_j]$ be the intersection of the expanded box of $r$, denoted $box_e(r)$ and the kernel function of $q$ along dimension $j$ (see Figure 4). We discretize the interval $[\lambda_j, \rho_j]$ by choosing a set of equidistant points $\lambda_j = \nu_1, \nu_2, \ldots, \nu_\alpha = \rho_j$ (where $\alpha$ is a sufficiently large constant), and update the value of $\Sigma_{rj}$ as follows:

$$\Sigma_{rj} = \Sigma_{rj} + \sum_{i=1}^{\alpha-1} ((\nu_i + \nu_{i+1})/2 - r_j)^2 wt_i,$$

where

$$wt_i = 0.75 \cdot \tau_q \frac{1}{B_{qj}} \int_{\nu_i}^{\nu_{i+1}} \left(1 - (\frac{x - q_j}{B_{qj}})^2\right) dx$$

is the (approximate) number of points of *leaf*$(q)$ whose $j$'th coordinate lies in the interval $[\nu_i, \nu_{i+1}]$. In other words, we approximate all points in this interval by its midpoint, and use this approximation to update the $\Sigma$ value for $r$.

The update procedures are summarized in Figure 5. The overall procedure for deleting a leaf is given in Figure 6 (a).

*Inserting a leaf.* Let $p$ be the point newly selected in the sample $S$, and $q$ be an existing sample point such that $p \in$ *leaf*$(q)$. We split *leaf*$(q)$ by a hyperplane passing through

PROCEDURE $\tau$-*Update*$(q, r, box(r))$
 $\tau_r$ += it sel(*box*$(r)$, *leaf*$(q)$);
**end**

PROCEDURE $\Sigma$-*Update*$(q, r, box(r))$
 *box*$_j(r)$: projection of *box*$(r)$ along dimension $j$;
 $\alpha$: constant;
 **for** $j = 1, \ldots, d$
  $[\lambda_j, \rho_j] = box_j(r) \cap [q_j - B_{qj}, q_j + B_{qj}]$;
  discretize $\lambda_j = \nu_1, \nu_2, \ldots, \nu_\alpha = \rho_j$;
  **for** $i = 1, \ldots, \alpha - 1$
   $wt_i = 0.75 \cdot \tau_q \frac{1}{B_{qj}} \int_{\nu_i}^{\nu_{i+1}} \left(1 - \left(\frac{x - q_j}{B_{qj}}\right)^2\right) dx$;
   $\Sigma_{rj}$ += $((\nu_i + \nu_{i+1})/2 - r_j)^2 wt_i$;
  **end for**
 **end for**
**end**

**Fig. 5.** Updating values $\tau$ and $\Sigma$

the midpoint $(p + q)/2$. The direction of the hyperplane is chosen according to the alternating rule of a kd-tree, i.e., if $i$ is the splitting dimension for the parent of $q$, then we split $q$ along dimension $(i + 1)$ mod $d$. Let *box*$_c(q)$ denote the contracted box that bounds the new *leaf*$(q)$. We set $\tau_q$ and $\tau_p$ to be the selectivity of *box*$_c(q)$, respectively *box*$(p)$, with respect to the set of points previously contained in *leaf*$(q)$ (in fact, $\tau_p$ is one more than the selectivity, to account for the new point $p$). We then compute the $\Sigma$ values for $p$ and $q$ using the procedure described in Figure 5, using *box*$_c(q)$ and *box*$(p)$ as the third argument. The overall procedure is summarized in Figure 6 (b).

Clearly, the update time is dominated by the leaf deletion procedure. Its time complexity is proportional to the number of leaves in the subtree rooted at the sibling of the leaf, which is $O(|S|)$ in the worst case. Hence, we conclude with the following.

**Theorem 2.** *The update and query time for our online range searching procedure is* $O(|S|)$*, where* $S$ *is the sample size.*

*Extension to turnstile model.* The overall approach can also be extended to handle the case when points are allowed to be deleted from the data stream. If the point $p$ to be deleted is not a kernel center, we compute $s_i$ such that $p \in$ *leaf*$(s_i)$. We then subtract 1 from $\tau_i$, and $(p_j - s_{ij})^2$ from $\Sigma_{ij},\ 1 \le j \le d$. If $p$ is a kernel center, then we delete *leaf*$(p)$ and choose the next point inserted in the stream to replace $p$ in the sample. This approach suffers from the same problems as maintaining a data sample over deletions, which is that we can no longer guarantee (with high probability) that the sample is uniform with respect to the data currently present in the stream. However, if the deletions do not exhibit strong spatial or temporal locality, then the approach is likely to work well in practice.

PROCEDURE *Delete*($leaf(q)$)
Compute $\mathcal{N}(q)$;
**for each** $r \in \mathcal{N}(q)$
$box_e(r)$: expanded box of $r$;
$\tau$-*Update*$(q, r, box_e(r))$;
$\Sigma$-*Update*$(q, r, box_e(r))$;
**end for**
Replace parent of *leaf*($q$) by sibling of *leaf*($q$) in $\mathcal{T}(\mathcal{S})$;
**end**

(a)

PROCEDURE *Insert*($leaf(p)$)
Compute $q$ so that $p \in$ *leaf*($q$);
Split *leaf*($q$) through $(p+q)/2$ along appropriate axis;
$box_c(q)$: contracted box of new *leaf*($q$);
$oldq$: all values assoc. with $q$ before splitting;
$\tau_p = \tau_q = 0$;
$\Sigma_{qj} = \Sigma_{pj} = 0, 1 \leq j \leq d$;
$\tau$-*Update*$(oldq, q, box_c(q))$;
$\tau$-*Update*$(oldq, p, box(p))$; $\tau_p += 1$;
$\Sigma$-*Update*$(oldq, q, box_c(q))$;
$\Sigma$-*Update*$(oldq, p, box(p))$;
**end**

(b)

**Fig. 6.** (a) Deleting *leaf*($q$). (b) Inserting *leaf*($p$)

## 4 Experimental Results

We evaluate the performance of our on-line algorithm on both synthetic and real data sets, in different number of dimensions and under varying query loads. We focus on range selectivity queries in the experiments described in this section, as they are one of the most common type of queries asked against multi-dimensional data, and have been extensively studied in previous literature. Note, however, that other types of queries can also be handled using the statistical information maintained by Local Kernels, and the range selectivity computation as a basic procedure. For example, the *rank* of a point $p = (p_1, \ldots, p_d)$, defined as the number of points dominated by $p$ on all coordinates, can be computed as the range selectivity of the hyper-rectangle $(-\infty, p_1) \times \cdots \times (-\infty, p_d)$. *Hot spots*, defined as unit cubes containing at least $\alpha n$ points in the stream, for some user-defined $0 < \alpha < 1$ (see [16]), could also be computed by answering range selectivity queries on an appropriate set of candidate cubes.

As we discuss below, the data distribution for one of the real datasets we used was significantly different from the synthetic data. However, our method returned good results on both distributions. There are two main issues we are interested in evaluating in our experimental set-up: the accuracy of our method with respect to existing techniques, both on-line and off-line, and the trade-off between accuracy and space usage.

We first provide experimental evidence that our proposed local-statistics kernels computation is competitive in an off-line setting with previous density-based methods. In fact, as we show below, it has better accuracy in most of the cases we study. This is an important issue, because the off-line accuracy of local kernels is an upper bound on the accuracy that we can achieve when moving to the on-line setting. Hence, we need to validate our strategy as a multi-pass method first. By this, we mean that instead of maintaining the approximate values for the number of points and standard distribution in a leaf, we compute them exactly in one pass over the current data, every time the set of kernel centers changes. Note that off-line local-statistics kernels are of independent

interest, as they can be used for summarization and mining for large-scale data warehousing. As we discuss below, in the context of data warehousing, off-line local kernels require only two data scans. Our experiments also show that they yield highly accurate results with small space requirements, even when query selectivity is very low.

Next, we study the performance of our on-line algorithm. We first compare it to the discrete one-pass histogram method proposed in [22], on a set of integer-valued two-dimensional data. Although the method can be used to answer range queries that arrive interspersed with the data, the expensive histogram computation makes it impractical for such a setting. Therefore, we restricted our experiments to the case when all queries arrive after the entire data stream has been processed. Finally, we evaluate our method in its most general setting, i.e. over real-valued data that arrives interspersed with the queries. We compare it with random sampling, which we consider as a base method for on-line range searching, due to its simplicity and effectiveness in practice. The version that we implemented uses reservoir sampling [24] to maintain a sample of the points in a streaming environment. We also consider the off-line version of local kernels computation, which we use as a basis of comparison for accuracy results. As mentioned above, this represents the best approximation that can be obtained via local kernels.

*Datasets and queries.* We used four datasets: a 2-dimensional synthetic set SD2, a 4-dimensional synthetic set SD4, and a 2-dimensional set NM2 containing network measurements. Each of the two synthetic datasets contains 1 million points, of which 90% are contained in clusters, and 10% are uniformly distributed. The data generator we used is similar to the one described in [2], which was introduced in order to model local dependencies in the data. There are 100 clusters in each dataset, and the points in each cluster are drawn from a normal distribution around a randomly chosen center. The variance in each cluster is determined randomly in the following manner: Fix a *spread parameter* $r$ and choose a scale factor $s_{ij} \in [1, s]$ uniformly at random, where $s$ is user defined. Then the variance of the normal distribution in cluster $i$ and along dimension $j$ is $(s_{ij} \cdot r)^2$. The number of points in each cluster is proportional to the realization of an exponential random variable. Once all clusters are generated, we compute a random permutation of the points, and choose that to be the order in which the data arrives. Thus, the stream does not exhibit temporal correlation, i.e. consecutive points are likely to belong to different clusters.

The network dataset NM2 contains 1 million two-dimensional data points with real-valued attributes. Each point is an aggregate of measurements taken during a fifteen-minute interval, reflecting minimum and maximum delay times between pairs of servers on AT&T's backbone network.[1] The dataset is only a small snapshot of the entire information stored in the course of a month, which we have chosen for the purpose of experimental evaluation. As mentioned in Section 1, summarization methods with small storage are highly desirable in this case, as older data is no longer stored on disk, making it difficult to access.

For every dataset, we create two query workloads. The queries are chosen randomly in the attribute space, but each query in a workload has approximately the same selectivity: 0.5% for the first workload, and 10% for the second. Hence, the first workload

[1] The proprietary nature of the data prohibits us from disclosing more details.

corresponds to low selectivity ranges, and reflects how much our kernel density approach manages to improve over random sampling. The second workload corresponds to high selectivity ranges, and is used in order to verify that our method does not result in an over-smoothing of the distribution function, which would then imply significantly under-estimating the query counts. All workloads contain 200 queries each. The timestamps of the queries are randomly interspersed in the data stream.

*Accuracy measure.* For the remainder of this paper, we report the accuracy of each method as the average 1-norm relative error, defined below. Because of the random nature of all the algorithms we discuss, each point on a graph is the average value over five runs.

For a query $Q_i = \langle i, R_i \rangle$, the relative estimation error of $Q_i$ is defined as

$$err_i = \frac{|sel(Q_i) - estimated\_sel(Q_i)|}{\max\{sel(Q_i), 1\}}.$$

Let $\{Q_{i1}, \ldots, Q_{ik}\}$ be the query workload for a given experiment. Then the average 1-norm error for this workload is defined as $avg\_err = (\sum_{j=1}^{k} err_{ij})/k$. Note that in the off-line setting, $N \leq i_1, \ldots, i_k$, where $N$ is the size of the data stream; i.e., all queries are asked after the entire data is seen.

*Validating local kernels in an off-line setting.* In the context of data warehousing, the computation of summaries or density functions does not take place concurrently with query processing. The assumption is that the entire data is available for pre-processing before queries can be answered. Hence, we do not need to maintain our local kernels under insertions and deletions from the sample set. Rather, we scan the data once to select our random sample of centers, compute the associated kd-tree, and then use a second scan to compute the exact values for the number of points and standard distribution in each leaf. All queries are answered after the two data scans are completed. We denote this algorithm by MPLKernels, from **M**ulti-**P**ass **L**ocal **Kernels**.

We will compare this approach with the kernel-density method proposed in [15], which we denote by GKernels (**G**lobal **Kernels**). The main difference between MPLKernels and GKernels is that the latter defines kernel bandwidths as functions of the global standard deviations of the data along each dimension. In their paper, the authors note that their algorithm can be implemented as a one-pass method, by approximating the global standard deviations with the standard deviations of the random sample. However, since we are interested in comparing the relative accuracy of the two methods, we report results for the two-pass GKernels, which provides better query estimates.

For the sake of completion, we also include accuracy results for the random sampling estimator, as well as for our on-line local kernels method, which we denote by LKernels. For the latter, we emphasize that, although we maintain the kernels in the on-line manner described in the previous section, the fact that all queries are processed after the entire data has been seen implies that only the last set of kernels is important. Hence, LKernels should be regarded, in this context, as a one-pass warehousing method, rather than an on-line one.

Figures 7 and 8 indicate that both MPLKernels and LKernels are competitive with GKernels in terms of accuracy. In fact, they out-perform it in almost all the cases, except

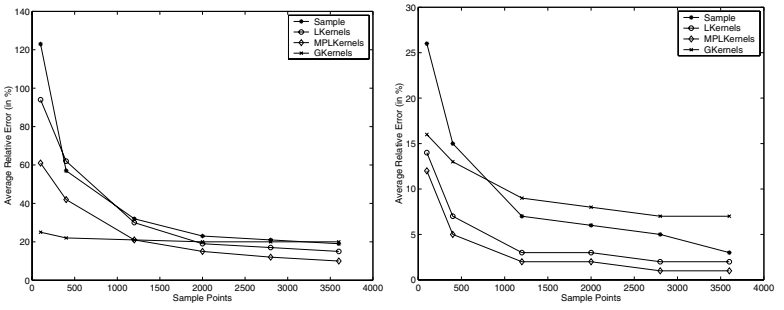


**Fig. 7.** Off-line query workloads on dataset SD2: (a) selectivity 0.5%; (b) selectivity 10%

for small (below 1500) sample sizes for SD2. This is not surprising, since intuitively, using local statistics should provide a better estimate on the data density. Gunopulos et al. [15] have also suggested that the performance of their density-based estimator would improve by using a clustering algorithm first, and then replacing global statistics by local statistics in each cluster. However, they did not provide any experimental results, and such an approach would be more difficult to adapt to an on-line setting. As noted in the previous section, one of the main advantages of using a kd-tree-like structure is that updating it is easy and fast.

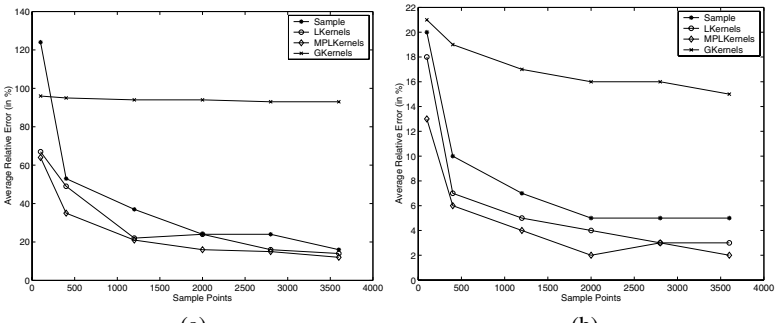


**Fig. 8.** Off-line query workloads on dataset SD4: (a) selectivity 0.5%; (b) selectivity 10%

Note that for the low selectivity workload, the accuracy of GKernels is almost independent of the sample size. This can be explained by the fact that using global statistics is too coarse an estimate of local density, and the accuracy degrades as the selectivity decreases. Perhaps even more surprising is the fact that the random sample estimator outperforms GKernels on both SD2 and SD4, for sample sizes bigger than 1000. In looking more closely at the results, we noticed that in these cases GKernels under-estimates almost all of the queries. This is due to an over-smoothing of the data, a problem that has been studied intensely in the statistics literature [20] in relation to density estimators. Our approach of computing local kernels significantly alleviates this drawback.

**Table 1.** The performance of the Histogram algorithm for various values of parameter $\rho$, compared with the performance of the LKernels algorithm

| | Error | Update time | Query time |
|---|---|---|---|
| LKernels | 35% | 0.0039ms | 2.5ms |
| Histogram (optimal) | 33% | 0.007ms | 2011ms |
| Histogram ($\rho = 3$) | 50% | 0.007ms | 286ms |
| Histogram ($\rho = 2$) | 60% | 0.007ms | 252ms |
| Histogram ($\rho = 1$) | 63% | 0.007ms | 211ms |
| Histogram ($\rho = 0.5$) | 81% | 0.007ms | 167ms |

*Comparison with histogram methods.* Histograms are a widely used method for computing summary statistics and answering selectivity queries. While off-line histogram computation has been studied for a long time, it is only more recently that the approach has been adapted to on-line settings. Two recent approaches have been proposed in [6,22]. The first algorithm, called STHoles [6], computes the histogram via a training phase, which poses queries and uses the exact answers to build the resulting histogram. The second approach [22] maintains a sketch of the actual data distribution, and uses it to extract histogram buckets. As the sketch can be maintained on-line, this method is closer in spirit to the problem we consider. Moreover, the experimental evaluation in [22] shows that it has comparable or sometimes better accuracy than STHoles. We therefore restrict our attention only to this algorithm.

We evaluate it on a synthetic two-dimensional dataset of 100x100, generated similarly to the one above, except that the points are restricted to the integer grid. For a fair comparison, we restrict both methods to use the same amount of memory cache, which is 11000 integers. This translates into 1200 sample points for Local Kernels (we use 9 integers per sample point). As for the dynamic histogram, a careful analysis of the results in [22] shows that the space utilization is $\min(n^2, sn) + 2s$, where $s$ is the sketch size, and $n$ is the attribute range (in our case, 100). Based also on the evaluations presented in the original paper, we use a sketch size of 500, and extract 50 buckets. We have implemented the faster heuristic (EGreedy), and present the accuracy and computation time in table 1. Update time represents the average time spent per data point, and query time is the average time to answer a query (it includes the histogram computation time for EGreedy). The parameter $\rho$ represents the ratio $\alpha/k$ in the original paper, which is used to speed up histogram computation, at the cost of reducing accuracy. The higher the value of $\rho$, the better the accuracy. The notation 'optimal' denotes the algorithm EGreedy in which the optimal bucket is computed at each iteration.

As is apparent from Table 1, our approach is at least two orders of magnitude faster than the on-line histograms method, as well as significantly more accurate in most cases. The only situation in which EGreedy is slightly more accurate is when optimal buckets are computed, but in that case EGreedy is three orders of magnitude slower. Note also that the update time per point for our method is about half the time required by EGreedy. Thus, in an on-line environment in which points arrive with reasonably high frequency, and/or the queries are interspersed with the points, Local Kernels is clearly a better choice. Even in a setting in which all queries are computed after the entire dataset has been seen, Local Kernels may still be the preferred solution, as the small accuracy gain of EGreedy comes at a much higher cost in terms of processing time. We conclude that,

although designed with enough flexibility to handle real-valued data, Local Kernels is competitive as an on-line algorithm over discrete data, as well.

*General on-line setting.* We now evaluate the accuracy of our method for real-valued datasets in an on-line context, in which queries arrive interleaved with points. Each query must be processed with respect to the data seen so far, rather than at the end. We first present the performance of LKernels for the 2 and 4-dimensional synthetic datasets, and then discuss the NM2 set. As mentioned above, we use both the random sample estimator and MPLKernels as basis of comparison for accuracy results. In this context, MPLKernels performs a data scan every time it must answer a query: it first computes the exact number of points and the standard deviations in each leaf of $\mathcal{T}(\mathcal{S})$, and then processes the query. Note that in this case MPLKernels is an impractical approach. We include it here only as a benchmark, to better understand the limits of our method.

As expected, MPLKernels has the smallest relative error for all sample sizes. However, it is important to note that, just as in the off-line setting, the error of LKernels is always smaller than that of random sampling, which shows that we are indeed able to minimize the problem of over-smoothing in a consistent manner.

It is interesting to look at the graphs corresponding to LKernels in Figures 9 and 7 (respectively, Figures 10 and 8) side by side. The relative errors are very similar,

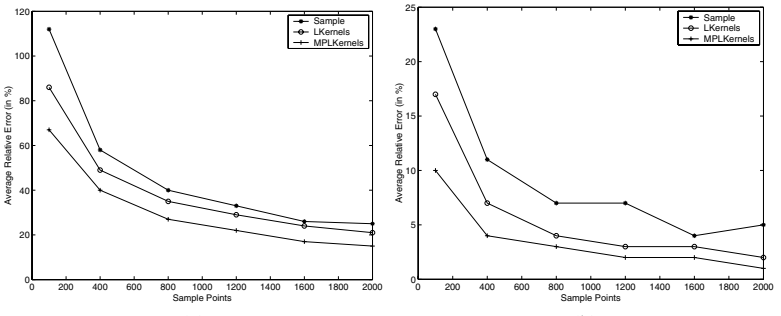


**Fig. 9.** Query workloads on dataset SD2: (a) selectivity 0.5%; (b) selectivity 10%

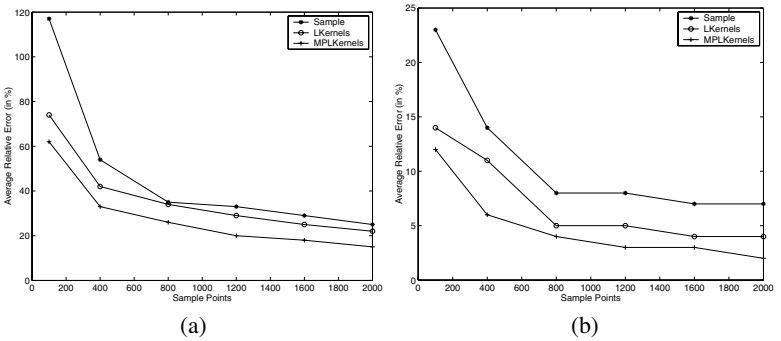


**Fig. 10.** Query workloads on dataset SD4: (a) selectivity 0.5%; (b) selectivity 10%

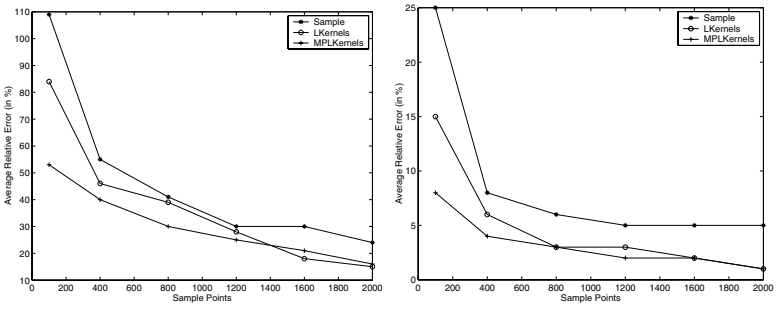


**Fig. 11.** Query workloads on dataset NM2: (a) selectivity 0.5%; (b) selectivity 10%

whether the queries are answered on the fly, or all at the end. This indicates that the performance of our method does not decrease with time, as we update our data structure and statistics estimates.

The experiments on the network dataset NM2 are shown in Figure 11. Unlike the synthetic data, for which we randomly chose the order in which points arrive, in this case points have a well established order given by the timestamp of the measurement. However, since each point refers to a different network connection, there is little locality in the data. The graphs exhibit the same patterns as before, with LKernels proving the robustness of the kernel-based methods.

## 5 Conclusions

In this paper we have proposed a new approach for computing density estimators over spatial data, and showed how it can be adapted to on-line environments. Our method maintains a random sample with a kd-tree-like structure on top of it, which permits the estimators to easily adapt to changes in the locality of the data. Given these density estimators, we can approximate the selectivity of range queries that arrive interspersed in the data stream. Our algorithm requires no a priori knowledge of the range of attribute values, nor of the number of tuples in the data stream, and is thus easy to use in a large number of practical applications. We have also provided extensive experimental evaluations that prove that the method is competitive (in terms of both accuracy and running time) with off-line summarization approaches and with one-pass histograms. Finally, we note that the idea of maintaining an indexing structure over spatial data streams, together with density functions, may be of independent interest, such as for visualizing the distribution of low-dimensional streaming data (e.g., network measurements).

## References

1. S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation n spatial databases. In *Proceedings of ACM SIGMOD*, pages 13–24, 1999.
2. C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *Proceedings of ACM SIGMOD*, pages 61–72, 1999.

3. N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of ACM PODS*, pages 10–20, 1999.
4. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the 28th Annu. ACM Symp. on the Theory of Computing (STOC)*, pages 20–29, 1996.
5. B. Blohsfeld, D. Korus, and B. Seeger. A comparison of selectivity estimators for range queries on metric attributes. In *Proceedings of ACM SIGMOD*, pages 239–250, 1999.
6. N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: a multidimensional workload-aware histogram. In *Proceedings of ACM SIGMOD*, pages 211–222, 2001.
7. G. Cormode and S. Muthukrishnan. The count-min sketch and its applications. In *Proceedings of LATIN*, pages 29–38, 2004.
8. N. A. C. Cressie. *Statistics for Spatial Data*. J. Wiley & Sons, New York, 1993.
9. A. Das, J. Gehrke, and M. Riedewald. Approximation techniques for spatial data. In *Proceedings of ACM SIGMOD*, pages 695–706, 2004.
10. A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the 34th Annu. ACM Symp. on the Theory of Computing (STOC)*, pages 389–398, 2002.
11. A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. One-pass wavelet decompositions of data streams. *IEEE Trans. Knowl. Data Eng.*, 15(3):541–554, 2003.
12. M. Greenwald and S. Khanna. Efficient online computation of quantile summaries. In *Proceedings of ACM SIGMOD*, pages 58–66, 2001.
13. S. Guha, C. Kim, and K. Shim. Xwave: Approximate extended wavelets for streaming data. In *Proceedings of the 30th VLDB Conference*, 2004.
14. S. Guha and N. Koudas. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In *Proceedings of ICDE*, pages 567–578, 2002.
15. D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *Proceedings of ACM SIGMOD*, pages 463–474, 2000.
16. J. Hershberger, N. Shrivastava, S. Suri, and C. D. Toth. Adaptive spatial partitioning for multidimensional data streams. In *Proceedings of ISAAC 2004 (LNCS 3341)*, pages 522–533, 2004.
17. G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *Proceedings of ACM SIGMOD*, pages 251–262, 1999.
18. J. Pach and P. K. Agarwal. *Combinatorial Geometry*. J. Wiley & Sons, New York, 1995.
19. V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd VLDB Conference*, pages 486–495, 1997.
20. D. W. Scott. *Multivariate Density Estimation*. Wiley-Interscience, 1992.
21. S. Suri, C. D. Toth, and Y. Zhou. Range counting over multidimensional data streams. In *Proceedings of Symp. on Computational Geometry (SCG)*, 2004.
22. N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *Proceedings of ACM SIGMOD*, pages 428–439, 2002.
23. V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.
24. J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.