Rectangle-Efficient Aggregation in Spatial Data Streams

Srikanta Tirthapura [*]
Iowa State University
snt@iastate.edu

David P. Woodruff
IBM Research-Almaden
dpwoodru@us.ibm.com

ABSTRACT

We consider the estimation of aggregates over a data stream of multidimensional axis-aligned rectangles. Rectangles are a basic primitive object in spatial databases, and efficient aggregation of rectangles is a fundamental task. The data stream model has emerged as a de facto model for processing massive databases in which the data resides in external memory or the cloud and is streamed through main memory. For a point p, let $n(p)$ denote the sum of the weights of all rectangles in the stream that contain p. We give near-optimal solutions for basic problems, including (1) the k-th frequency moment $F_k = \sum_{\text{points } p} |n(p)|^k$, (2) the counting version of stabbing queries, which seeks an estimate of $n(p)$ given p, and (3) identification of heavy-hitters, i.e., points p for which $n(p)$ is large. An important special case of F_k is F_0, which corresponds to the volume of the union of the rectangles. This is a celebrated problem in computational geometry known as "Klee's measure problem", and our work yields the first solution in the streaming model for dimensions greater than one.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

General Terms

Algorithms, Theory

Keywords

rectangle efficiency, distinct elements, frequency moments, streaming, spatial databases, data mining

1. INTRODUCTION

Spatial and temporal data arise in diverse domains such as geographic information systems, astronomy, VLSI design, and animation. Quoting Guting [28], a spatial database such as OpenGIS [1] "needs to deal with large collections of relatively simple geometric objects". Perhaps the most basic type of object in spatial and spatiotemporal databases is the multidimensional rectangle. For instance, [11] proposes modeling of coordinates of objects in space and time through "parametric rectangles", which are multi-dimensional axis-aligned rectangles, formed through a cross product of intervals. In the constraint database model [34] each object is itself formed by the intersection of constraints and is often an axis-aligned rectangle, and query processing in a constraint database can be viewed as a computation over this set of geometric objects; for example, as studied in [6, 46]. Aggregation of axis-aligned rectangles is also considered in the context of Online Analytical Processing (OLAP) [35, 42, 48, 51].

We consider aggregation of multi dimensional rectangles in the standard one-pass data stream model using limited workspace. In addition to online aggregation of a massive stream that cannot be physically stored, a data stream algorithm is useful for processing queries on a large database stored in external memory because it can be implemented using a memory-efficient sequential scan through the data. When new items arrive, the algorithm can update its state without having to rescan the entire dataset. While the data stream model has been widely studied over the past decade or so, there has been little work on aggregating high dimensional geometric objects.

1.1 Problem Definition

We consider a discrete d-dimensional universe $[\Delta]^d = \{1, 2, \ldots, \Delta\}^d$, where Δ is the maximum coordinate along any dimension. While assuming the universe is discrete is not very common in computational geometry, where typically the coordinates are assumed to be real, finite precision is a common assumption in the data stream literature. Without such an assumption, storage is not well-defined, since, e.g., many real numbers can be encoded into a single real number. Geometric problems in the data stream literature are thus often studied with finite precision, see [30]. Also, by imposing a sufficiently fine grid and snapping the input points to grid points, it is easy to adapt our algorithms to the case when the minimum inter-point distance is 1 and the diameter is bounded by Δ.

The input is a stream Υ of m items, where each item is a rectangle and an integer (positive or negative) weight associated with it:

$$\Upsilon = \{(r_1, w_1), (r_2, w_2), \cdots, (r_m, w_m)\},$$

[*]Supported in part by NSF grants 0834743, 0831903.

where $r_i \subseteq [\Delta]^d$ and $w_i \in \mathbb{Z}$. For a point p, its frequency $n(p)$ is defined as the sum of the weights $\sum_{(r_i,w_i)\in\Upsilon, p\in r_i} w_i$ of all rectangles that contain p.

We consider the problem of estimating the kth frequency moment $F_k(\Upsilon)$, defined as $F_k(\Upsilon) = \sum_{\text{points } p} |n(p)|^k$. When each input rectangle is a single point, this definition reduces to the classical definition of the kth frequency moment of a stream, which has been very well studied in the literature starting from the work of Alon, Matias, and Szegedy [2] [1] (see for example the references in the recent papers [3, 9]). As shown by Braverman and Ostrovsky [10], techniques for understanding the frequency moments were used to characterize the class of all sketchable functions. Note that $F_0(\Upsilon)$ is the volume of the union of all rectangles in the input stream, and is a celebrated problem in computational geometry, known as Klee's measure problem [33].

We also consider one of the most famous problems in spatial databases - the stabbing problem (see, e.g., [46]). Given a point p and an aggregation function F, a query reports the aggregate of the weights of the rectangles in Υ containing p. Typically F is MAX or SUM. See [35, 42, 48, 51] for numerous applications of such queries in spatial databases. Note that a data structure for stabbing queries when F is SUM can also be used to compute the maximum depth $\max_p n(p)$. The points p for which $n(p)$ is large are often called "heavy hitters" or "iceberg queries".

Despite the large body of work on streaming algorithms, the streaming complexity of aggregates on spatial data is not well understood. Classical streaming algorithms can be used to process rectangles by considering the points in the rectangle one at a time. Such solutions, while space-efficient, have prohibitively high time complexity since the time taken to process a rectangle is proportional to its volume. The first works to try to overcome this were for 1-dimensional rectangles, i.e., line segments. The notion of a *range-efficient* sketch was introduced in [5] for F_k-estimation, and further refined for F_0 estimation in [43, 47], allowing one to process a segment in time only logarithmic in its length. Many other problems have been reduced to range-efficient F_k-estimation, for $k \geq 0$, such as distinct summation problem [18, 43], duplicate insensitive sketches [38], maximum-dominance norm [19], self-join size of the symmetric difference of relations [44], and counting triangles in graphs [5]. Attempts at generalizing this to larger dimensions fall short of what is desired. One generalization is in [5], where statistics of d-dimensional points were estimated by designing *range-efficient algorithms in every coordinate*. However, this is still essentially a one-dimensional object, since the coordinates in the object are allowed to have a range of values along one dimension, but fixed along the other dimensions. If we were to use a range-efficient algorithm to process a two dimensional rectangle, the time taken would still be proportional to the length of the smaller side.

1.2 Our Results

We give the first *rectangle-efficient* algorithms for computing frequency moments and identifying heavy hitters in the data stream model. By rectangle-efficient, we mean that our algorithms process each rectangle in the stream in time significantly sublinear in the volume of the rectangle.

[1]See Google Scholar for a list of the 900+ citations of their paper on frequency moments.

We say that a random variable x is an (ε, δ)-approximation to a number y if $\Pr[|x - y| \leq \varepsilon y] \geq 1 - \delta$. We let $O^*(f)$ denote a function of the form $f \cdot (\varepsilon^{-1} d \log(m\Delta/\delta))^{O(1)}$.

1.2.1 *Frequency Moments*

We present the following results for $F_k, k \geq 0$.

- For F_0, i.e., the Klee's measure problem, we give an algorithm that returns an (ε, δ)-approximation to the volume of the union of rectangles in Υ using space as well as update time $O^*(1)$. This is the first efficient solution to Klee's measure problem in the streaming model for $d \geq 2$. In the turnstile model (which allows negatively-weighted rectangles [37]), this is the first algorithm for F_0 even for $d = 1$.

- For $F_k, 0 < k \leq 2$, we give (ϵ, δ)-approximation algorithms using space and time $O^*(1)$ in the turnstile model. This is most interesting for F_1 with deletions, which, e.g., can be used to measure variation distance of distributions on multi-dimensional data approximated by box histograms [49]. We note that in the special case of $d = 1$ and $k = 2$, a range-efficient algorithm for F_2 was previously known [25].

- For F_k, $k > 2$, our space bound is polynomial in Δ^d, namely, it is $O^*(\Delta^{d(1-2/k)})$, while our update time is also $O^*(\Delta^{d(1-2/k)})$. The dependence on Δ in the space complexity is the best possible, due to known lower bounds even when the input consists only of points [4]. That is, a special case of our problem is when all of the input rectangles are points. Given a universe of m possible items, the known space lower bound [4] is $\Omega(m^{1-2/k})$, and plugging in $m = \Delta^d$ establishes an $\Omega(\Delta^{d(1-2/k)})$ bound. We note this is first algorithm to achieve optimal space (up to $O^*(1)$ factors) with less than the trivial Δ^d running time, even for $d = 1$.

  For the insertion only model, we present an alternate algorithm for $F_k, k > 2$ that uses $O^*(\Delta^{d(1-1/k)}) \cdot O(\log \Delta)^d$ space, which is slightly more (for small d) but still sublinear, but reduces the time per rectangle to $O^*(1) \cdot O(\log \Delta)^d$.

  Rectangle-efficient algorithms were not known for any $F_k, k > 2$, for $d > 1$. Even for $d = 1$ algorithms for $F_k, k > 2$ were known only in the insertion only model [5], and used space and time $O^*(\Delta^{1-1/k})$. Our algorithms improve space or time, work in the general turnstile model, and handle $d > 1$.

1.2.2 *Stabbing Queries, Maximum Depth, and Heavy Hitters*

We present a data structure that uses $O^*(1)$ space and $O^*(1)$ time to process each rectangle, and can return for any $p \in [\Delta]^d$, an estimator of $n'(p)$ of $n(p)$ (the "stabbing number" of p) such that with high probability $n'(p) \in \left[n(p) \pm O\left(\varepsilon\sqrt{\sum_{q \neq p} n(q)^2}\right)\right]$. It is well-known that in general it is impossible to get a better additive approximation to $n(p)$ in $O^*(1)$ streaming space (see, e.g., [4]).

- **Maximum Depth.** Clearly such a data structure gives an additive $O(\varepsilon\sqrt{\sum_{q \neq p} n(q)^2})$ approximation to

$\max_p n(p)$, also known the *maximum depth* [14] of a set of axis-aligned rectangles.

- **Heavy Hitters.** The heavy-hitters problem asks to return all points p for which $n(p)$ is at least $\alpha\sqrt{\sum_q n(q)^2}$, but not return any point p for which $n(p)$ is less than $(\alpha-\varepsilon)\sqrt{\sum_q n(q)^2}$ (note that this is a stronger guarantee than returning those p for which $n(p) \geq \alpha\sqrt{\sum_q n(q)^2}$). Given any algorithm for estimating $n(p)$, there is a generic procedure which, with an additional $O^*(1)$ factor in space and time, converts it into an algorithm for finding the heavy hitters. The idea is to build a quadtree of the input grid, obtaining $\log\Delta$ levels, and in each region in each level, the weight of all points in the region is aggregated into a single "meta-point". In each level we estimate $n(q)$ with additive error, where q is a meta-point, and we recurse on those q for which $n(q)$ is large. See, e.g., section 5.2 "Turnstile Case" in [20]. We thus find all heavy hitters in space and time $O^*(\varepsilon^{-1})$.

We note that many of our results have space and time complexity polynomial in the dimension d. While many of our applications are for constant d, this is an important feature for large values of d.

Extensions: As mentioned, many of our results allow for arbitrary positive or negative weights on the input rectangles, and thus allow us to estimate F_k of the difference of two data streams. Thus, for example, we can rectangle-efficiently approximate the volume of the difference of two collections of rectangles, and find the heavy hitters in the presence of negative weights. The main application of negative weights is to analyzing the difference of two streams. This cannot be done using random sampling, for example. Our method can be applied to streams of other objects that can be approximated by a union of $O^*(1)$ axis-aligned rectangles. Rectangles are commonly used as bounding boxes for approximating other objects. See, e.g, [22] for ways of approximating a convex polygon this way.

1.3 Our Techniques

Many problems in the data stream literature can be solved by a combination of three techniques: (1) sub-sampling the items into a logarithmic number ϕ of levels, where in the j-th level, roughly a 2^{-j} fraction of items are included, (2) multiplying item weights by random signs, and (3) for each level of sub-sampling, hashing the included items into buckets and maintaining the sum of item weights in each bucket. Let us associate the points x in the input grid $[\Delta]^d$ with vectors in the vector space $\{GF(\Delta)\}^d$, where we assume Δ is a power of 2, and $GF(\Delta)$ denotes the finite field containing Δ elements. In the rest of this paper, we use $GF(\Delta)^d$ to denote $\{GF(\Delta)\}^d$. Each of these three operations can be thought of as applying a hash function to x. For (1), x is included in the j-th substream if $g(x) \leq \Delta^d/2^{j-1}$ for a random function $g : GF(\Delta)^d \to GF(\Delta^d)$, and an arbitrary mapping from $GF(\Delta)^d$ to the integers $\{1, 2, \ldots, \Delta^d\}$. For (2), the sign of x is equal to $s(x)$, where $s : GF(\Delta)^d \to \{-1, 1\}$ is a random function. For (3), the bucket x is assigned to is $f(x)$ for a random function $f : GF(\Delta)^d \to GF(B)$, where B is the total number of buckets (w.l.o.g., B is a power of 2).

Our first insight is that if we were lucky enough that each of g, s, and f could be implemented by a pairwise-independent hash function for the problem we are trying to solve, then we would be in great shape. Indeed, it is well-known that the function mapping x to $Ax+b$, for a random matrix A and vector b with entries in $GF(\Delta)$ is pairwise-independent. Hence, the set of input points x which map to the j-th level of sub-sampling, have a positive sign, and map to a particular bucket $k \in GF(B)$ is just the intersection of three equations: $A^1x + b^1 = 0$, $A^2x + b^2 = 0$, and $A^3x+b^3 = k$, for appropriately chosen A^1, A^2, A^3, b^1, b^2, and b^3. By using Gaussian elimination, which is efficient for d-dimensional x, we can equivalently describe such x as the solutions to a single equation $Cx = d$ for a certain matrix C and vector x. We thus obtain $\phi \cdot 2 \cdot B$ systems of equations, one for each combination of level of subsampling, positive or negative sign, and bucket k. Typically ϕ and B are $O^*(1)$, and so we can afford to spend time solving each such system.

To solve each system given an input rectangle, there are several approaches possible. One is to decompose the rectangle into dyadic intervals along each dimension, and note that the cross product of dyadic intervals over the d dimensions forms a partition of the input rectangle into smaller rectangles of a special kind. Importantly, for each such smaller rectangle, it corresponds to a set of vectors x for which some number of coordinates take on fixed values, while the remaining coordinates range over all possible bitstrings as x ranges over the rectangle. This property allows us to count the number of solutions x using Gaussian elimination, and since the number of rectangles in the partition is not too large (for small d), we can sum up the number of solutions over the rectangles in the partition.

While multi-dimensional dyadic decomposition is possible, it leads to bounds that are exponential in d. Although many of the applications are for constant d, it is still useful to have bounds which are polynomial in d. To achieve this, one can use a technique of Pagh [41] instead of multidimensional dyadic decomposition. In this case, s and f must have a special form, namely if $x = (x_1, \ldots, x_d) \in GF(\Delta)^d$ then $s(x) = s_1(x_1) \cdots s_d(x_d)$ and $f(x) = f(x_1) + \cdots + f(x_d) \bmod B$, for independent functions $s_i : GF(\Delta) \to \{-1, 1\}$ and $f_i : GF(\Delta) \to GF(B)$, with an arbitrary mapping of elements of $GF(B)$ to elements of $\{0, 1, 2, \ldots, B-1\}$. Here each s_i and f_i is drawn from a pairwise-independent family of functions. Then, if the rectangle R has the form $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$, then using the Fast Fourier Transform we can efficiently compute for each $k \in GF(\Delta)$, the quantity $\sum_{x \in R \mid f(x)=k} s(x)$, given only $\sum_{x_j \in [a_j, b_j] \mid f_j(x_j)=k} s_j(x_j)$ for each $j \in [d]$. The latter d quantities can be found using 1-dimensional dyadic decomposition and solving systems of linear equations, as described above.

One problem with this overall approach is that while it was recently discovered that g and f can be implemented with pairwise-independent hash functions [3, 9] for estimating F_k, it was not known that s can also be implemented with a pairwise-independent hash function. For example, the seminal work of Alon, Matias, and Szegedy [2] requires 4-wise independence for estimating F_2 to within a constant factor. Intuitively, this is because the algorithm takes the sum of squares of counters as its expectation, which requires 2-wise independence for sign-cancellation, and then takes the square of this to bound the variance, which requires 4-wise independence for sign-cancellation. The previous algorithm for F_k with the smallest independence known which followed this three-step process (sub-sampling, random signs, bucketing) was 4-wise independence [9], and this is because their algorithm effectively estimates F_2 inside of a routine for estimating the heavy hitters (the so-called CountSketch algorithm) so as to make sure that all heavy hitters returned have relative-error estimates. We observe that if we relax the guarantee of the estimates to the heavy hitters to give total additive error, rather than a per heavy hitter relative error guarantee, then we can slightly tweak the analysis of [9] to show that correctness is maintained. Now we do not need to implicitly estimate F_2, and since CountSketch only requires pairwise independence, there is an algorithm for estimating F_k using this three step process, for any $k \geq 0$, with only pairwise independence.

It is also likely to be possible to use our ideas to rectangle-efficiently implement a recent F_k-estimation algorithm of [3], which achieves pairwise-independence (they do not achieve range or rectangle-efficiency). There the authors skip the sub-sampling phase by multiplying the items by appropriate weights (so-called "precisions"). We would need to discretize their weights and enumerate them (as we are enumerating levels of sub-sampling above) and apply our procedure for counting the number of items in an input rectangle with a given weight. This may be less efficient, depending on the discretization required. We choose to follow the exposition of [9] both for simplicity and due to its extreme generality: as we shall see in Section 3.1, we can obtain a rectangle-efficient algorithm for any problem for which there is a rectangle-efficient heavy hitters algorithm.

1.4 Other Approaches

One could ask why 4-wise independence for s would not suffice for our purposes. The difficulty is that it does not seem easy to compose with the 2-wise independent functions f and g. An example 4-wise independent function would be $s(x) = x^T Ax + Bx + c$ for certain random matrices A, B, and c from a special family of second-order Reed-Muller codes [26]. We do not know how to quickly count the number of solutions x to the equation $s(x) = d$ for general matrices A, B and vector c, while we can count such x for heavily-structured A, B, and c coming from a special family based on Reed-Muller codes. This is problematic when composing it with the linear constraints imposed by f and g, since the new matrices obtained after substitution no longer correspond to a function s in the special family. There is some work on range-summable functions for higher-order Reed-Muller codes [12], but as far as we are aware the problem for order three and above is #P-complete [36].

Another completely different approach, often used in computational geometry, would be to treat multidimensional rectangles as points in a higher dimension (typically twice the original dimension), and to convert operations on rectangles to corresponding operations on higher dimensional points. This works for certain types of operations on rectangles. For example, given a collection $\mathcal{R}$ of line segments (i.e. one dimensional rectangles) and a query segment r^*, suppose we want to find all segments $r \in \mathcal{R}$ such that r completely contains r^*. For segment $[a, b]$ the function $\mathcal{P}()$ yields a point in two dimensions, defined as $\mathcal{P}([a, b]) = (-a, b)$. It is easy to see that segment r_1 contains segment r_2 iff $\mathcal{P}(r_1)$ dominates $\mathcal{P}(r_2)$, i.e., each coordinate of $\mathcal{P}(r_1)$ is greater than or equal to the corresponding coordinate of $\mathcal{P}(r_2)$. Thus, the problem of finding all 1-dimensional rectangles that contain a query (1-dimensional) rectangle can be posed as the problem of finding all 2-dimensional points that dominate a given 2-dimensional point. This reduction can be easily generalized to higher dimensions. While the above reduction works for certain relationships among a pair of rectangles, it does not seem to work for the aggregate functions that we consider. For example, it is not clear how the computation of the size of the union of rectangles can be reduced to a computation over a set of higher dimensional points, and if such a reduction can be easily used.

1.5 Related Work

Das, Gehrke, and Riedewald [21] consider approximation techniques for spatial data streams, and present algorithms for estimating aggregates on data, using an extension of the linear AMS sketches (Alon, Matias, Szegedy [2]) for F_2. Their algorithm is for rectangle intersection, and does not give sublinear space guarantees in the worst case. Hershberger, Shrivastava, and Suri [29] consider the sketching of a stream of points in 2 dimensions, to maintain its geometric shapes through a collection of convex hulls. Note that their input is a stream of (multidimensional) points, while our input is a stream of rectangles. There has been a long line of work on Klee's measure problem [33, 23, 7, 40, 13, 16, 17, 24, 50], but none of these works seem relevant for computation in the streaming model. There is a previous attempt at a solution to Klee's measure problem on a data stream [45]. This was found to have an error, and was withdrawn from arXiv. This work uses a different technique than was used in [45]. The algorithm in [45] was based on sampling, along the lines of [43], while our algorithm is based on sub-sampling and linear sketches with special properties.

Klee [33] introduced the one-dimensional version of the problem now known as Klee's measure problem (KMP) in the RAM model, and gave an $O(m \log m)$ time algorithm for computing the size of the union of m line segments in one dimension. This was shown to be time-optimal (for a class of algorithms) by Fredman and Weide [23]. Bentley [7] presented an $O(m \log m)$ time and $O(m)$ space solution for KMP on m two dimensional rectangles. Bentley's algorithm can be extended to d dimensions, but takes time $O(m^{d-1} \log m)$ time. Overmars and Yap [40] gave an algorithm that improved the time for d dimensions to $O(m^{d/2} \log m)$. More recent work includes [13, 16, 17, 24, 50]. Recent work has focused on the space complexity of the algorithm. Chen and Chan [16] gave an algorithm for KMP in two dimensions that uses $O(m^{3/2} \log m)$ time with $O(\sqrt{m})$ extra space, which is in addition to the space needed for representing the input rectangles (so this does not apply to the data stream model). Note that the time taken by this algorithm is worse than the optimal time achieved by Bentley, but the (additional) space required is small. Vahrenhold [50] improved the (additional) space to $O(1)$, while maintaining the same running time. In the model used by [16, 50], there was no constraint on the number of passes of the processor.

Paper Overview: We present our main primitive data structure RectangleCountSketch in Section 2. We show how to use this to estimate F_k rectangle-efficiently for any $k \geq 0$ in Section 3. In Section 4, we present the rectangle-efficient algorithm for F_k for insertion-only streams.

2. RECTANGLE-COUNTSKETCH

We define RectangleCountSketch(γ) for an input parameter $\gamma \in (0,1)$. We assume all points in the input grid $[\Delta]^d$ are identified with the vector space $GF(\Delta)^d$, where $GF(\Delta)$ denotes the finite field with Δ elements, where we assume Δ is a power of 2.

Notice that since Δ is a power of 2, we can also think of elements of $GF(\Delta)$ as length-($\log \Delta$) bitstrings where addition of $x, y \in GF(\Delta)$ corresponds to component-wise XOR of the corresponding bitstrings. Similarly, we can think of elements in the vector space $GF(\Delta)^d$ as length-($d \log \Delta$) bitstrings where addition of $x, y \in GF(\Delta)^d$ corresponds to component-wise XOR of the corresponding bitstrings.

The goal of the algorithm is to provide an estimate $\Phi(x)$ to v_x for each $x \in GF(\Delta)^d$, where v_x is the current sum of the weights of the rectangles in the input stream which contain x. Our estimate $\Phi(x)$ will be correct up to an additive error that can be reduced by increasing the memory requirements of the scheme.

The algorithm is the same as CountSketch [15] except that it is also rectangle-efficient. Let $B > 3\gamma^{-2}$ be a power of 2. We assume $B \leq \Delta$.

Fact 1. *([27]) Let $f : GF(2)^k \to GF(2)^r$ be defined as $f(x) = Ax + b$, where A is a uniformly random $k \times r$ matrix and b is a uniformly random column vector of length r. Then f is a pairwise-independent function, that is, for any $x \neq x' \in GF(2)^k$ and $y, y' \in GF(2)^r$,*

$$\Pr[f(x) = y \wedge f(x') = y'] = \frac{1}{2^{2r}}.$$

Notice that pairwise-independence also implies that for any fixed $x \in GF(2)^k$, $f(x)$ is uniformly random over the choice of f.

In fact, it is known that f is pairwise independent even if A is chosen to be a uniformly random Toeplitz matrix and b is a uniformly random vector. This latter property can be used to reduce the time complexity of computing Ax by using the Fast Fourier Transform (FFT) to perform the matrix-vector multiplication. Moreover, the memory required to represent A is smaller since a random Toeplitz matrix requires less randomness to specify. Since these optimizations are relatively minor in our setting, as k and r will be quite small, we omit them in our analysis.

Proof. See page 8 of [27] for the construction with Toeplitz matrices. See Appendix 2 of [27] for the proof for both random matrices A and Toeplitz matrices A. ■

Remark 2. The fact that A can be a uniformly random Toeplitz matrix and b a uniformly random vector can be used to reduce the time complexity of computing Ax by using the Fast Fourier Transform (FFT) to perform the matrix-vector multiplication. It can also be used to reduce the space complexity since a random Toeplitz matrix requires less randomness to generate. Since we will use matrices A with only a logarithmic number of rows and columns, such improvements are relatively minor and we omit them in our algorithm and analysis below.

Define pairwise-independent functions:

- For each $j \in [d]$, independently choose $f_j : GF(\Delta) \to GF(B)$, $f_j(x) = A_j \cdot x + b_j$ for a random $(\log \Delta) \times (\log B)$ matrix A_j with entries in $GF(2)$, and random $b_j \in GF(B)$. Here x is interpreted as a length-($\log \Delta$) bitstring and b_j is interpreted as a length-($\log B$) bitstring.
- For each $j \in [d]$, independently choose $s_j : GF(\Delta) \to \{-1, 1\}$, $s_j(x) = (-1)^{\langle \sigma_j, x\rangle + \tau_j}$, for random vector $\sigma_j \in GF(\Delta)$ and $\tau_j \in GF(2)$. Here x and σ_j are interpreted as length-($\log \Delta$) bitstrings, and $\langle \sigma_j, x \rangle$ denotes their inner product.

Now let $x = (x_1, \ldots, x_d) \in GF(\Delta)^d$ and define the following functions:

- $f(x) = f_1(x_1) + f_2(x_2) + \cdots + f_d(x_d) \mod B$.
- $s(x) = s_1(x_1) \cdot s_2(x_2) \cdots s_d(x_d)$.

It is easy to verify that $f : GF(\Delta)^d \to GF(B)$ and $s : GF(\Delta)^d \to \{-1, 1\}$ are pairwise independent functions since $f_1, \ldots, f_d$, $s_1, \ldots, s_d$ are each pairwise independent, and independent of each other.

The basic data structure maps an input vector $v \in \mathbb{Z}^{[\Delta]^d}$ to a vector $c \in \mathbb{Z}^{\{0,1,2,\ldots,B-1\}}$ of "counters", where for each "bucket" $k \in \{0, 1, 2, \ldots, B-1\}$, we have counter

$$c_k = \sum_{\{x \ \mid \ f(x) = k\}} s(x) \cdot v_x.$$

Here and throughout, we often arbitrarily associate the buckets, indexed by elements of $GF(B)$, with the integers $0, 1, 2, \ldots, B-1$. Observe that this data structure maintains a linear function of v, which allows us to process updates to the coordinates of v. We will show how to process updates to rectangles of coordinates of v very efficiently.

For each $x \in GF(\Delta)^d$, we estimate v_x as:

$$\Phi(x) = s(x) \cdot c_{f(x)}$$

Lemma 3. *For any $\eta > 0$,*

$$\Pr[|\Phi(x) - v_x| > \eta \|v\|_2] \leq \frac{1}{\eta^2 B}$$

Proof. For the expectation, we use pairwise-independence of s and that $\mathbf{E}[s(y)] = 0$ for any vector y:

$$\begin{aligned}
\mathbf{E}[\Phi(x)] &= \mathbf{E}\left[s(x) \sum_{y \ \mid \ f(x)=f(y)} s(y) v_y\right] \\
&= v_x + \sum_{y \mid f(x)=f(y), \ x \neq y} \mathbf{E}[s(x)s(y)] \cdot v_y \\
&= v_x
\end{aligned}$$

For the second moment, we additionally use the pairwise-independence of f:

$$\begin{aligned}
\mathbf{E}[\Phi(x)^2] &= \mathbf{E}\left[c_{f(x)}^2\right] = \sum_{y, y' \ \mid \ f(x)=f(y)=f(y')} \mathbf{E}[s(y)s(y')] v_y v_{y'} \\
&= \sum_{y \ \mid \ f(x)=f(y)} v_y^2 \leq v_x^2 + \frac{1}{B} \sum_{x \neq y} v_y^2 \\
&\leq v_x^2 + \frac{\|v\|_2^2}{B}
\end{aligned}$$

Hence, $\mathbf{Var}[\Phi(x)] \leq \frac{\|v\|_2^2}{B}$. By Chebyshev's inequality,

$$\Pr[|\Phi(x) - v_x| > \eta \|v\|_2] \leq \frac{\mathbf{Var}[\Phi(x)]}{\eta^2 \|v\|_2^2} \leq \frac{\|v\|_2^2}{\eta^2 B \|v\|_2^2} = \frac{1}{\eta^2 B}.$$

■

By Lemma 3, for $B > \frac{3}{\gamma^2}$, we have $\Pr[|\Phi(x) - v_x| \leq \gamma \|v\|_2] \geq \frac{2}{3}$. Hence, if we take $r = O(d \log(\Delta/\delta))$ independent copies of the basic data structure, obtaining $\Phi^1(x), \ldots, \Phi^r(x)$ for each $x \in GF(\Delta)^d$, and set $\hat{\Phi}(x) = \mathrm{median}_{\ell=1}^{r} \Phi^\ell(x)$, then by a Chernoff and union bound, with probability at least $1-\delta$, for all $x \in GF(\Delta)^d$, $|\hat{\Phi}(x) - v_x| \leq \gamma \|v\|_2$. The total space complexity of the scheme for $B = O(\gamma^{-2})$, ignoring the cost to represent the hash functions, is $O(\gamma^{-2} \cdot d \log(\Delta/\delta))$ words, since this is the number of counters maintained. Notice that each f_j and s_j requires only $O(\log \Delta \cdot \log B)$ bits to store in each of the $O(d \log(\Delta/\delta))$ basic data structures, giving a space bound of $O(d^2 \log(\Delta/\delta) \log \Delta \log B)$ to represent all of the hash functions.

Rectangle-Efficiency: Given a rectangle $R = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$, we show how to efficiently update our data structure. We show how to do this independently for each basic data structure, and multiply the time by $O(d \log(\Delta/\delta))$. To implement our basic data structure, we show how to update our counters given R.

Since our data structure is linear, it suffices to show how to perform updates of the form $(R, 1)$, i.e., a rectangle update with weight 1. Indeed, if we show how to compute a vector c_{change} of changes to the counters, i.e., a vector for which $c \leftarrow c + c_{change}$ to process an update of the form $(R, 1)$, then to process an update of the form (R, w) for a general weight w, it suffices to set $c \leftarrow c + w \cdot c_{change}$.

For each $j \in [d]$, we recursively partition $[a_j, b_j]$ into $t_j = O(\log \Delta)$ intervals:

$$[a_{j,1}, b_{j,1}], [a_{j,2}, b_{j,2}], \ldots, [a_{j,t_j}, b_{j,t_j}].$$

We first find the largest integer q so that there is an integer u with $[u2^q, (u+1)2^q) \subseteq [a_j, b_j]$. The important property of an interval of the form $[u2^q, (u+1)2^q)$ is that all $i \in [a_j, b_j]$ have the same length-$(N-q)$ prefix α in their binary representation, while the suffix ranges over all 2^q possible bit strings. We then recursively divide $[a_j, u2^q - 1]$ and $[(u+1)2^q, b_j]$. This is a standard decomposition into dyadic intervals (see, e.g., [5]), and the recursion produces $t_j = O(\log \Delta)$ disjoint intervals.

For each $j \in [d]$, each dyadic interval $[a_{j,\ell}, b_{j,\ell}]$ for an $\ell \in [t_j]$, and each bucket $k \in \{0, 1, \ldots, B-1\}$, we show how to find the number of $x \in [a_{j,\ell}, b_{j,\ell}]$ for which $f_j(x) = k$ and $s_j(x) = 1$, and the number of $x \in [a_{j,\ell}, b_{j,\ell}]$ for which $f_j(x) = k$ and $s_j(x) = -1$.

If $x \in [a_{j,\ell}, b_{j,\ell}]$ satisfies $f_j(x) = k$ and $s_j(x) = 1$, then $A_j x + b_j = k$ and $\langle \sigma_j, x \rangle + \tau_j = 0$. For $x \in GF(\Delta)$, we break each x into the concatenation of x^{pre} and x^{suf} as follows. Suppose $[a_{j,\ell_j}, b_{j,\ell_j}]$ is a dyadic interval of the form $[u2^q, (u+1)2^q)$ for integers u and q. We let x^{pre} be the common length $(\log \Delta) - q$ prefix of all $x \in [a_{j,\ell_j}, b_{j,\ell_j}]$, and we let x^{suf} be the suffix of x, which ranges over all 2^q bitstrings as x ranges over the interval $[a_{j,\ell_j}, b_{j,\ell_j}]$. We append q 0s to x^{pre} and we prepend $(\log \Delta) - q$ 0s to x^{suf}, and so we have $x = x^{pre} + x^{suf}$.

We break the matrix A_j into two contiguous groups A_j^{pre} and A_j^{suf} of its columns, so that

$$A_j x \;=\; A_j^{pre} x^{pre} + A_j^{suf} x^{suf},$$

and similarly break σ into two contiguous groups σ^{pre} and σ^{suf} of its columns so that

$$\langle \sigma_j, x \rangle \;=\; \langle \sigma_j^{pre}, x^{pre} \rangle + \langle \sigma_j^{suf}, x^{suf} \rangle$$

Using that x^{pre} is a fixed vector, we can write the constraint $A_j x + b_j = k$ as

$$A_j^{suf} x^{suf} = -A_j^{pre} x^{pre} - b_j + k,$$

where the right hand side is a fixed vector, and the vector x^{suf} is unconstrained. Similarly, we can write the constraint $\langle \sigma_j, x \rangle = \tau_j$ as

$$\langle \sigma_j^{suf}, x^{suf} \rangle = -\langle \sigma_j^{pre}, x^{pre} \rangle + \tau_j,$$

where the right hand side is a fixed bit, and the vector x^{suf} is unconstrained.

We can thus write the conjunction of these linear systems as another linear system, and use Gaussian elimination to diagonalize the system to count the number of solutions x^{suf} to the conjunction of these linear systems in $O^*(d^3) = O^*(1)$ time. We solve such a linear system for each $\ell \in [t_j]$, each $k \in \{0, 1, \ldots, B-1\}$, and for both $s_j(x) = 1$ and $s_j(x) = -1$. Since the $[a_{j,\ell}, b_{j,\ell}]$ partition $[a_j, b_j]$, we have thus found, for each $k \in \{0, 1, \ldots, B-1\}$, the total number $e_{j,k,1}$ of $x \in [a_j, b_j]$ for which $f_j(x) = k$ and $s_j(x) = 1$, as well as the total number $e_{j,k,-1}$ of $x \in [a_j, b_j]$ for which $f_j(x) = k$ and $s_j(x) = -1$. We do this for each $j \in [d]$. The total time spent is therefore $B \cdot \mathrm{poly}(d) = \gamma^{-2} \cdot O^*(1)$.

We now leverage a technique due to Pagh [41]. Pagh shows how to efficiently combine a 1-dimensional CountSketch applied to a set S_1 with a 1-dimensional CountSketch applied to a set S_2, to obtain a 2-dimensional CountSketch applied to the set $S_1 \times S_2$. We can this technique with our technique of computing a 1-dimensional CountSketch, described above.

Pagh's idea is to associate the B counters of CountSketch with the coefficients of a univariate polynomial with formal variable z. If the current state of the counters is $c_0, c_1, \ldots, c_{B-1}$, then a polynomial that represents this state is $\sum_{k=0}^{B-1} c_k z^k$. Given the counters, this polynomial can be constructed in $O(k)$ time, and given this polynomial, all of the counters can be extracted in $O(k)$ time.

In our setting, for each $j \in [d]$ we create a corresponding univariate degree-$(B-1)$ polynomial $p_j(z)$. The coefficient of z^k is equal to $e_{j,k,1} - e_{j,k,-1}$. That is,

$$p_j(z) = \sum_{k=0}^{B-1} \sum_{x \in [a_j, b_j] \text{ such that } f_j(x) = k} s_j(x) \cdot z^k.$$

We would like to compute the following polynomial $q(z)$:

$$\begin{aligned} q(z) &= \sum_{\substack{k\in\{0,\ldots,B-1\}\\ x\in R | f(x)=k}} s(x)\cdot z^k \\ &= \sum_{\substack{k\in\{0,\ldots,B-1\}\\ x\in R | f(x)=k}} s(x_1)\cdots s(x_d)\cdot z^{f(x_1)+\cdots+f(x_d) \bmod B} \\ &= \sum_{\substack{k\in\{0,\ldots,B-1\}\\ x\in R | f(x)=k}} s(x_1)\cdots s(x_d)\cdot z^{f(x_1)}\cdots z^{f(x_d)} \\ &\qquad \bmod (z^B-1) \\ &= \prod_{j=1}^{d} \sum_{x_j\in[a_j,b_j]} s(x_j) z^{f(x_j)} \bmod (z^B-1) \\ &= \prod_{j=1}^{d} p_j(z) \bmod (z^B-1), \end{aligned}$$

where for univariate polynomials $p(z)$ and $q(z)$ with rational coefficients, the notation $p(z) \bmod q(z)$ indicates the unique polynomial (the "remainder") $r(z)$ for which $p(z) = m(z)\cdot q(z) + r(z)$, where $r(z)$ has degree strictly less than that of $q(z)$.

Lemma 4. *For any two polynomials $p_1(z)$ and $p_2(z)$, we have $(p_1(z)\cdot p_2(z)) \bmod (z^B-1) = ((p_1(z) \bmod (z^B-1))\cdot (p_2(z) \bmod (z^B-1))) \bmod (z^B-1)$.*

Proof. Let $r(z)$ be the remainder of division of $p_1(z)\cdot p_2(z)$ by z^B-1, so that $p_1(z)p_2(z) = m(z)(z^B-1)+r(z)$ with the degree of $r(z)$ less than B, for some polynomial $m(z)$. Let $r_1(z)$ be the remainder of division of $p_1(z)$ by z^B-1, so $p_1(z) = m_1(z)(z^B-1)+r_1(z)$ with the degree of $r_1(z)$ less than B, for some polynomial $m_1(z)$. Similarly define $r_2(z)$ and $m_2(z)$. Then $p_1(z)p_2(z) \bmod (z^B-1) = (m_1(z)m_2(z)(z^B-1)^2 + m_1(z)(z^B-1)r_2(z) + m_2(z)(z^B-1)r_1(z) + r_1(z)r_2(z)) \bmod (z^B-1) = r_1(z)\cdot r_2(z) \bmod (z^B-1)$, which completes the proof. ■

It follows by Lemma 4 that to compute $q(z)$ we can first compute $p_1(z)\cdot p_2(z) \bmod (z^B-1)$, then compute the product of this polynomial with $p_3(z)$, and then take this modulo z^B-1, etc. In this way, the computation of $q(z)$ is that of d multiplications of polynomials of degree $B-1$. Each polynomial multiplication can be done in $O(B\log B)$ time using the FFT, provided that B is a power of 2. It is also trivial to reduce the polynomial modulo z^B-1, since this just involves replacing a monomial z^k with $z^{k \bmod B}$. Hence, the total time to compute $q(z)$ given $p_1(z),\ldots,p_d(z)$ is $O(dB\log B)$.

Given $q(z)$, we can read off its coefficients to update each of the counters c_k in $O(B)$ time. Hence, the total update time is $\gamma^{-2}\cdot O^*(1)$. We summarize our findings in the following theorem.

Theorem 5. *The data structure* RectangleCountSketch(γ) *can be updated rectangle-efficiently in time $\gamma^{-2}\cdot O^*(1)$. The total space is $\gamma^{-2}\cdot O^*(1)$ words. The data structure can be used to answer any query $x \in GF(\Delta)^d$, returning a number $\Phi(x)$ with $|\Phi(x)-v_x| \le \gamma\|v\|_2$. The algorithm succeeds on all queries simultaneously with probability $\ge 1-\delta$.*

3. RECTANGLE-EFFICIENT F_K

In this section we consider rectangle-efficient estimation of F_k, for constant $k \ge 0$. Here, F_k of a vector $v \in [\Delta]^d$ is defined as $\sum_{j\in[\Delta]^d} v_j^k = \|v\|_k^k$, which is equal to the k-th power of the k-norm [2]. Note that 0^0 is interpreted as 0 in case of F_0. On page 8 of [9], Braverman and Ostrovsky provide an algorithm RecursiveSum[0](D,ε) which takes in a stream D, an error parameter ε, and provides an $(\varepsilon, 3/10)$-approximation to the k-th frequency moment F_k, for constant $k \ge 2$. The success probability can be amplified to $1-\delta$ by independent repetition. We will build on this work in several ways. Their algorithm is in turn a simplification of earlier work [8, 32].

We choose to follow [9] since it provides a simpler exposition and has several properties we will exploit. First, we will show that it works for any constant real $k \ge 0$ instead of just $k \ge 2$. This first part is a simple, yet very useful observation. Indeed, most algorithms in the literature for estimating F_k, $k \le 2$, rely on k-stable random variables [31], and we do not know how to make them rectangle-efficient. Next, and importantly, we relax a requirement of their analysis in Lemma 6 below, which is needed in order for us to combine their algorithm with our RectangleCountSketch algorithm.

Let $\phi = O(d\log\Delta)$. The algorithm of Braverman and Ostrovsky (in our language) chooses a pairwise-independent hash function $g : GF(\Delta)^d \to GF(\Delta^d)$ and defines ϕ substreams $D_1,\ldots,D_\phi$, where the j-th substream D_j consists of the input stream D restricted to those items $i \in GF(\Delta)^d$ for which $g(i) \le \Delta^d/2^{j-1}$, where we arbitrarily map the elements of $GF(\Delta^d)$ to the elements of the set $\{1,2,\ldots,\Delta^d\}$. For each substream, the algorithm runs the CountSketch algorithm of Charikar, Chen, and Farach-Colton [15]. The CountSketch algorithm of [15] is identical to our RectangleCountSketch algorithm when all stream updates are single points, rather than rectangles, and additionally requires that $d=1$. From this data structure, at the end of the stream Braverman and Ostrovsky produce an estimate to F_k, $k \ge 2$. This is described in more detail in Figure 1 in the next subsection.

In our implementation we simply replace CountSketch with RectangleCountSketch. This completely specifies the algorithm up to the parameter γ we choose in RectangleCountSketch, and how we process the data structure to output an estimate to F_k after seeing the data stream.

The following lemma is a relaxed version of Corollary 5.3 in [9], and yields a procedure to return the heavy hitters in a specific substream D_j from our RectangleCountSketch data structure after processing the stream. We return a set P of items together with approximations to their frequencies. The difference between this Lemma and [9] is that we bound the total additive error of the approximations while [9] guarantees that the frequency of each item in P has a small relative error. The reason for this is that we do not know how to achieve a small relative error for each item by using only pairwise independence, which is in turn crucial for making our algorithm rectangle-efficient.

Lemma 6. *Fix a value $j \in [\phi]$. Let $v \in \mathbb{Z}^{[\Delta]^d}$ be the frequency vector of items $i \in [\Delta]^d$ in substream D_j, that is $v(i)$ is the number of occurrences of i in D if $g(i) \le \Delta^d/2^{j-1}$*

[2]Technically, for $k<1$, $\|v\|_k$ is not a norm since it doesn't satisfy the triangle inequality, but it is still a well-defined quantity.

(under an arbitrary mapping of $GF(\Delta)^d$ to $\{1, 2, \ldots, \Delta^d\}$), and is 0 *otherwise. For any constant real number $k \in (0, \infty)$, there is an algorithm which uses* RectangleCountSketch(γ) *with $\gamma = \Theta(\varepsilon^{1+1/k}\alpha^{1/k})$ if $k \in (0, 2]$, and $\gamma = \Theta(\varepsilon^{1+1/k}\alpha^{1/k}/\Delta^{d/2-d/k})$ for $k > 2$, so that with probability at least $1-\delta$, it outputs a set P of $O(1/\alpha)$ pairs (i, v_i') for which $\sum_{(i,v_i')\in P} ||v_i'|^k - |v_i|^k| \le \varepsilon\|v\|_k^k$, and such that all elements i with $|v_i|^k \ge \alpha\|v\|_k^k$ appear as the first element of some pair in P. The algorithm uses $O^*(\gamma^{-2})$ words of space.*

PROOF. By Theorem 5, with probability at least $1-\delta$, for all i we have $|\Phi(i)-v_i| \le \gamma\|v\|_2$ if we use $O(\gamma^{-2} \cdot d\log(\Delta/\delta))$ words of space in RectangleCountSketch. We assume this event occurs and add δ to the error probability. Our algorithm uses the output of RectangleCountSketch(γ) to find the set P of the pairs $(i, \Phi(i))$ corresponding to the i with the largest $2/\alpha$ values $\Phi(i)$.

We will call an i for which $|v_i| \ge (\varepsilon\alpha/4)^{1/k}\|v\|_k$ *heavy*. Otherwise, we will say i is *light*.

The following fact is well-known and follows from basic inequalities on norms.

FACT 7. *If $k \ge 2$, then $\|v\|_2 \le \Delta^{d/2-d/k}\|v\|_k$, while if $k < 2$, then $\|v\|_k \ge \|v\|_2$.*

Since k is an absolute constant, for any $(i, \Phi(i)) \in P$ for which $|v_i| \ge (\varepsilon\alpha/4)^{1/k}\|v\|_k$ we have that (for appropriately chosen constants in the $\Theta(\cdot)$ notation defining γ above):

$$\begin{aligned}\left(1 - \frac{\varepsilon}{10k}\right)|v_i| &\le |v_i| - \frac{\gamma\cdot\max(1,\Delta^{d/2-d/k})}{(\varepsilon\alpha/4)^{1/k}}|v_i| \\ &\le |v_i| - \gamma\cdot\max(1,\Delta^{d/2-d/k})\|v\|_k \\ &\le |v_i| - \gamma\|v\|_2 \le |\Phi(i)|,\end{aligned}$$

where we have used Fact 7. We also have that, using Fact 7,

$$\begin{aligned}|\Phi(i)| &\le |v_i| + \gamma\|v\|_2 \\ &\le |v_i| + \gamma\cdot\max(1,\Delta^{d/2-d/k})\|v\|_k \\ &\le |v_i|(1+\gamma\cdot\max(1,\Delta^{d/2-d/k})/(\varepsilon\alpha/4)^{1/k}) \\ &\le (1+\varepsilon/(10k))|v_i|.\end{aligned}$$

Hence, for such i, we have

$$(1-\varepsilon/3)|v_i|^k \le |\Phi(i)|^k \le (1+\varepsilon/3)|v_i|^k.$$

Next, for any $(i, \Phi(i)) \in P$ for which $|v_i| < (\varepsilon\alpha/4)^{1/k}\|v\|_k$, we have, using Fact 7,

$$\begin{aligned}0 &\le |\Phi(i)|^k \le (|v_i| + \gamma\|v\|_2)^k \\ &\le \|v\|_k^k((\varepsilon\alpha/4)^{1/k} + \gamma\|v\|_2/\|v\|_k)^k \\ &\le \|v\|_k^k((\varepsilon\alpha/4)^{1/k} + \Theta(\varepsilon^{1+1/k}\alpha^{1/k})) \\ &\le \|v\|_k^k\cdot\varepsilon\alpha/3.\end{aligned}$$

Hence,

$$\begin{aligned}&\sum_{(i,\Phi(i))\in P} ||\Phi(i)|^k - |v_i|^k| \\ =& \sum_{\text{heavy } i} ||\Phi(i)|^k - |v_i|^k| + \sum_{\text{light } i} ||\Phi(i)|^k - |v_i|^k| \\ \le& \frac{\varepsilon}{3}\sum_{\text{heavy } i}|v_i|^k + \frac{2}{\alpha}\cdot\max\left\{\frac{\varepsilon\alpha\|v\|_k^k}{4}, \frac{\varepsilon\alpha\|v\|_k^k}{3}\right\} \le \varepsilon\|v\|_k^k.\end{aligned}$$

Finally, let T be the set of i for which $|v_i|^k \ge \alpha\|v\|_k^k$. It remains to show each $i \in T$ occurs in some pair in P. As shown above, for such i, we have $|\Phi(i)|^k \ge (1-\varepsilon/3)|v_i|^k$. If i is not in a pair in P, then there are at least $2/\alpha$ different j for which

$$|\Phi(j)|^k \ge |\Phi(i)|^k \ge (1-\varepsilon/3)|v_i|^k \ge (1-\varepsilon/3)\alpha\|v\|_k^k.$$

It follows that j cannot be light, since for such j we have $|\Phi(j)|^k \le (\varepsilon\alpha/3)\|v\|_k^k$. On the other hand, for a heavy j, we have $|\Phi(j)|^k \le (1+\varepsilon/3)|v_j|^k$, and so there would need to be at least $2/\alpha$ different heavy j for which $|v_j|^k \ge \frac{1}{1+\varepsilon/3}\cdot(1-\varepsilon)\alpha\|v\|_k^k$, which is a contradiction, as the sum of v_j^k over such j would be larger than $\|v\|_k^k$. ■

We next prove the analogue of Lemma 6 in the case that $k = 0$.

LEMMA 8. *Fix a value $j \in [\phi]$. Let $v \in \mathbb{Z}^{[\Delta]^d}$ be the frequency vector of items $i \in [\Delta]^d$ in substream D_j, that is v_i is the number of occurrences of i in D if $g(i) \le \Delta^d/2^{j-1}$ (under an arbitrary mapping of $GF(\Delta)^d$ to $\{1, 2, \ldots, \Delta^d\}$), and is* 0 *otherwise. There is an algorithm which uses* RectangleCountSketch(γ) *with $\gamma = \Theta(\alpha\varepsilon)$ so that with probability at least $1-\delta$, it outputs a set P of $O(1/\alpha)$ items i for which $\sum_{i\in P}|1 - |v_i|^0| \le \varepsilon\|v\|_0$, and such that all elements i with $|v_i| \ge \alpha\|v\|_0$ occur in P. The algorithm uses $O^*(\gamma^{-2})$ words of space.*

PROOF. The algorithm is to run RectangleCountSketch with parameter $\gamma = \alpha\varepsilon/3$. Then, find the set P of items i for which $\Phi(i) \ge 2/3$. If $|P| \le 1/\alpha$, then output P. Otherwise, output $\emptyset$.

First suppose that $|v|_0 \le 1/(\varepsilon\alpha)$. Then by Theorem 5, with probability at least $1-\delta$, for $\gamma = \alpha\varepsilon/3$ and for all i, we have $|\Phi(i) - v_i| \le 1/3$. This implies that the output of RectangleCountSketch(γ) can be used to find the set of at most $1/(\varepsilon\alpha)$ non-zero values v_i. Hence, it can output this set as P if $|P| \le 1/\alpha$ and incur zero additive error. If the set $|P|$ it finds satisfies $|P| > 1/\alpha$, then it outputs $\emptyset$, and incurs zero additive error. In both cases, all i with $|v_i| \ge \alpha\|v\|_0$ are output, as desired.

Now suppose that $\|v\|_0 > 1/(\varepsilon\alpha)$. Since $|P| \le 1/\alpha$ and $\varepsilon\|v\|_0 > 1/\alpha$, and since there is no i with $|v_i| \ge \alpha\|v\|_0$, the lemma is trivially satisfied. ■

The key is that our Lemma 6 and Lemma 8 can be used instead of Corollary 5.3 of [9] in the remaining analysis of [9], replacing various steps that require that each $|v_i'| = (1\pm\varepsilon)|v_i|$ for i occurring as a first coordinate of a pair in P with the weaker condition of Lemma 6 that instead bounds the total additive error (and similarly for Lemma 8). We give the details of the tweaked analysis in Section 3.1.

Rectangle-Efficiency: Given a rectangle $R = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$, we show how to efficiently update the data structure used in RecursiveSum[0](D, ε).

The idea is the same as in our RectangleCountSketch algorithm. Namely, for each substream D_j, for each bucket $k \in [B]$ in a basic data structure of RectangleCountSketch, we need to find the number of $i \in R$ which are placed in bucket k with a sign of -1, and the number of i with a sign of $+1$. Since g is a pairwise-independent hash function, it can be expressed as $Ai+b$, where $A \in GF(\Delta)^d \times GF(\Delta)^d$ and b is a vector of length Δ^d. Then an element i is in D_j if and only if $Ai + b$ has a prefix of at least $j-1$ zeros. Equivalently, by removing all but $j-1$ rows of A and b, obtaining matrix A' and vector b', we require that $A'i + b' = 0$. As in the RectangleCountSketch algorithm, we can count the number of solutions to the intersection of this linear equation and the linear equations imposed by the pairwise-independent hash functions h and s used in RectangleCountSketch in the previous section. We thus obtain the following theorem, whose proof is given in Subsection 3.1.

THEOREM 9. *For constant $k \in [0, \infty)$, there is a rectangle-efficient single-pass streaming algorithm which we call* RecursiveSum[0](D, ε) *which outputs an (ε, δ)-approximation to F_k. For $k \in [0, 2]$, it uses $O^*(1)$ words of space and $O^*(1)$ time to process each rectangle in the stream. For $k > 2$, it uses $O^*(\Delta^{d-2d/k})$ words of space and $O^*(\Delta^{d-2d/k})$ time to process each rectangle in the stream.*

3.1 Finishing the Analysis for F_k-Estimation

We show how to use our Lemma 6 and Lemma 8 in place of Corollary 5 in the analysis of [9] to produce an estimate to F_k, $k \geq 0$, thereby filling in the details of the proof of Theorem 9. We point out where our relaxation to total additive error is used. The analysis closely follows that of [9].

We recall the following definitions of [9]. Let V be an N-dimensional vector of non-negative values V_j (we note our notation is slightly different than that of [9], who use lower-case v_j to denote the coordinate values) and $\|V\|_1 = \sum_{j=1}^{n} V_j$ be its 1-norm. An element V_i is α-major with respect to V if $V_i \geq \alpha \|V\|_1$. A set $S \subseteq [N]$ is an α-core with respect to V if $i \in S$ for any α-major V_i.

Here we relax the definition of (α, ε)-cover in [9]. The first bullet below is where we incorporate our total additive error guarantee. In [9], the first bullet below instead states that for all $j \in [t]$, $(1-\varepsilon)V_{i_j} \leq w_j \leq (1+\varepsilon)V_{i_j}$.

A non-empty set Q of the form $\{(i_1, w_1), \ldots, (i_t, w_t)\}$ for some $t \in [n]$ and distinct $i_1, \ldots, i_t$ is a **relaxed** (α, ε)-cover with respect to V if the following hold:

1. $\sum_{j \in [t]} |V_{i_j} - w_j| \leq \varepsilon \|V\|_1$.
2. $\forall i \in [n]$, if V_i is α-major then there exists a $j \in [t]$ such that $i_j = i$.

For a distribution $\mathcal{D}$ on sets Q of the form $\{(i_1, w_1), \ldots, (i_t, w_t)\}$, we say that $\mathcal{D}$ is δ-good if

$$\Pr_{Q \sim \Delta}[Q \text{ is a relaxed } (\alpha, \varepsilon) - \text{cover}] \geq 1 - \delta.$$

We sometimes define a randomized mapping g from vectors V to sets Q and say that g is δ-good with respect to V if its output distribution is δ-good. For a set $Q = \{(i_1, w_1), \ldots, (i_t, w_t)\}$ of the above form, let

$$Ind(Q) = \{i_1, \ldots, i_t\}.$$

For $i \in Ind(Q)$, let $w_Q(i)$ be such that $(i, w_Q(i)) \in Q$.

LEMMA 10. *(Corollary 3.6 of [9]) Let g be a δ-good mapping for a vector V and let $Q \sim g(V)$. Let H be a random N-dimensional vector independent of V and Q with pairwise-independent zero-one entries H_i. Define*

$$X' = \sum_{i \in Ind(Q)} V_i + 2 \sum_{i \notin Ind(Q)} H_i V_i.$$

Then $\Pr[|X' - \|V\|_1| \geq \varepsilon \|V\|_1] \leq \frac{\alpha}{\varepsilon^2}$. (we note there is a typo in their paper, they write $\frac{\varepsilon}{\alpha^2}$ instead of the correct $\frac{\alpha}{\varepsilon^2}$.)

Let $H^1, \ldots, H^\phi$ be i.i.d. random vectors with pairwise-independent zero-one entries. For two vectors U and V of dimension N, let $Had(V, U)$ be a vector of dimension N with entries $U_i \cdot V_i$. Define: $V^0 = V$ and $V^j = Had(V^{j-1}, H^j)$ for $j \in [\phi]$ (again, our notation is slightly different than that of [10], who use V_j to indicate what we are calling V^j). Let $Q^0 \sim V^0, Q^1 \sim V^1, \ldots, Q^\phi \sim V^\phi$. Define $w_j(i) = w_{Q^j}(i)$. Define the sequence:

$$X'_j = \sum_{i \in Ind(Q^j)} V_i^j + 2 \sum_{i \notin Ind(Q^j)} H_i^{j+1} V_i^j,$$

for $j = 0, 1, 2, \ldots, \phi - 1$, and $X'_\phi = \|V^\phi\|_1$.

FACT 11. *(Fact 3.7 of [9].)* $\Pr[\cup_{j=0}^{\phi} |X'_j - \|V^j\|| \geq \varepsilon \|V^j\|_1] \leq (\phi + 1)\left(\frac{\alpha}{\varepsilon^2} + \delta\right)$.

Define Y'_ϕ to be any random variable depending on V^ϕ for which $\Pr[|Y'_\phi - \|V^\phi\|_1| \geq \varepsilon \|V^\phi\|_1] \leq \delta$. For $j = 0, 1, 2, \ldots, \phi - 1$, define

$$Y'_j = 2Y'_{j+1} + \sum_{i \in Ind(Q^j)} (1 - 2H_i^{j+1}) w_i^j.$$

In the next lemma we use our relaxed (α, ε)-covers instead of the (α, ε)-covers in [9].

LEMMA 12. *For any ϕ, γ, and V, for $\alpha = \Theta(\gamma^2/\phi^3)$ and $\delta = \Theta(1/\phi)$, $\Pr[|Y'_0 - \|V\|_1| \geq \gamma \|V\|_1] \leq 1/5$.*

PROOF. The proof is almost identical to that of Lemma 3.8 of [9]. The only difference is that our Q^j are relaxed (α, ε)-covers, while their Q^j are (α, ε)-covers. The only place that changes is that in the end of the proof they argue $\Pr[\cup_{j=0}^{\phi} (|Err_j^3| \geq \varepsilon \|V^j\|_1)] \leq (\phi + 1)\delta$, using that $Err_j^3 = \sum_{i \in Ind(Q^j)} |w_j(i) - V_i^j|$ and their Q^j is an (α, ε)-cover. However, this bound also holds by definition for Q^j a relaxed (α, ε)-cover, and so their proof remains unchanged. ■

The next theorem follows from Lemma 12.

THEOREM 13. *(Theorem 4.2 of [9]) Algorithm* RecursiveSum *computes a $(1 \pm \varepsilon)$-approximation of $\|V\|_1$ and errs with probability at most $3/10$. The space complexity is $O(\log n)$ times that of finding a relaxed (α, ε)-cover with error probability $O(1/\log n)$, where α is set to $\varepsilon^2/\log^3 n$.*

To conclude the analysis, we simply plug in our Lemma 6 and Lemma 8 for computing a relaxed (α, ε)-cover for $k > 0$ and $k = 0$, respectively, into step 2 of RecursiveSum.

Proof (: of Theorem 9) We set the dimension N in the vectors V^j in RecursiveSum to equal Δ^d, and identify the coordinates with elements of $GF(\Delta)^d$. We define the vector V^0 as follows: $V_x^0 = |v_x|^k$, where v is the input vector and we are trying to estimate $\|v\|_k^k$. We need to show how to efficiently obtain a relaxed (α, ε)-cover for each V^j, where $\alpha = \varepsilon^2/\log n$. This is exactly what is given by Lemma 6 and Lemma 8. The time and space bounds follow by these lemmas, and the correctness follows from Theorem 13. ■

1. Generate $\phi = O(\log n)$ pairwise-independent zero one-vectors $H^1, \ldots, H^{\phi}$, and let D_j be the substream restricted to those inputs i with $H_i^1 \cdot H_i^2 \cdots H_i^j = 1$.
2. Let Q^j be a relaxed (α, ε)-cover for the vector V^j underlying D_j.
3. If $F_0(V^{\phi}) > 10^{10}$, then output 0 and stop. Otherwise compute $Y_{\phi} = \|V^{\phi}\|_1$.
4. For each $j = \phi - 1, \ldots, 0$, compute $Y_j = 2Y_{j+1} - \sum_{i \in Ind(Q^j)} (1 - 2H_i^j) w_i^j$.
5. Output Y_0.

Figure 1: The RecursiveSum[0](D, ε) algorithm of [9], with our relaxation in step 2 to relaxed (α, ε)-covers.

4. RECTANGLE-EFFICIENT $F_K, K > 2$, IN POLYLOGARITHMIC TIME IN THE INSERTION-ONLY MODEL

In [5] it was shown how to range-efficiently implement the algorithm of [2] in one dimension in the insertion-only model, retaining $O^*(\Delta^{1-1/k})$ bits of space, but reducing the time per range from $O^*(\Delta)$ to $O^*(\Delta^{1-1/k})$. Here we will considerably strengthen this for small values of d, improving the time per range from $O^*(\Delta^{1-1/k})$ to $O^*(1) \cdot O(\log \Delta)^d$, while retaining $O^*(\Delta^{1-1/k}) \cdot O(\log \Delta)^d$ bits of space. Also, our algorithm generalizes to yield a rectangle-efficient algorithm for $d > 1$:

Theorem 14. *There is a rectangle-efficient algorithm for positively-weighted input rectangles which (ε, δ)-approximates F_k, $k \geq 2$, using $O^*(\Delta^{d(1-1/k)}) \cdot O(\log \Delta)^d$ bits of space and $O^*(1) \cdot O(\log \Delta)^d$ time to process each rectangle.*

Proof. We use the F_k-estimation algorithm of [2]. On a stream of points, the algorithm of [2] works by sampling $r = O^*(\Delta^{d(1-1/k)})$ random positions $i_1, \ldots, i_r$ in the data stream, obtaining a list of points $p_{i_1}, \ldots, p_{i_r}$ occurring at these positions in the stream. Then for $j = 1, 2, \ldots, r$, the algorithm counts the number $f(p_{i_j})$ of occurrences of point p_{i_j} in the stream after position i_j. Note that if the points are weighted (positively) instead of just occurring with weight 1, and if W is the sum of their weights, then $i_1, \ldots, i_r$ would be chosen randomly from the set $\{1, 2, 3, \ldots, W\}$, and p_{i_j} would equal the point in the stream for which the sum of weights of points seen so far is equal to i_j. A natural implementation of this algorithm would be, for each input rectangle, to check how many points $p_{i_1}, \ldots, p_{i_r}$ occur in the rectangle, and update the counts $f(p_{i_j})$. This takes $O(r)$ time, and corresponds to the algorithm of [5].

We first show how to achieve amortized $O^*(1) \cdot O(\log \Delta)^d$ time. The idea is to batch updates of r rectangles together and build a data structure for the d-dimensional stabbing counting query problem on these r rectangles. More precisely, we partition the input stream into contiguous blocks of r rectangles, except for the last block which may contain fewer than r rectangles. To process a rectangle which does not occur at the end of a block, we simply store the endpoints of the rectangle. At the end of each block, we need to update the $f(p_{i_j})$ values (some of these may previously be 0). The naïve implementation would take $O^*(r^2)$ time. Instead, we use the following, which boils down to computing segment trees inside internal nodes of segment trees, nested d levels deep.

Theorem 15. *(See Theorem 4.2 in [39] for the result for $d = 1$. The extension to $d > 1$ is implied by the first two paragraphs of Section 4.2 of [39].) Given a set of n rectangles in $[\Delta]^d$, they can be stored in a multi-dimensional segment tree using $O(n \log^d n)$ words of space, which can be constructed in $O(n \log^d n)$ time, and such that for any point $p \in [\Delta]^d$, we can determine the sum of the weights of rectangles containing point p in $O(\log^d n)$ time.*

Using Theorem 15, we can build a multi-dimensional segment tree in $O(r \log^d r)$ time. Then, for each of the at most r points $p_{i_1}, \ldots, p_{i_r}$, we can determine its weight with a query which takes only $O(\log^d r)$ time, giving total time $O(r \log^d r)$ time for updating the $f(p_{i_j})$ values. The total space complexity of the scheme is $O^*(r \log^d r)$.

Finally, the algorithm can be de-amortized in the following way. Namely, since we spend $O(r \log^d r)$ time at the end of each block, we can instead spread the work required of one block over the updates of the next block. Hence, the time we take per input rectangle is $O^*(1) \cdot O(\log \Delta)^d$ in the worst-case. At the end of the stream, to return an answer we take $O^*(r \log^d r)$ time, both to combine the estimates $f(p_{i_j})$ for $j = 1, \ldots, r$, and to process the second to last and last blocks, whose work has been pushed to the end of the stream. ■

Acknowledgment: We would like to thank the anonymous referees for many helpful comments, especially those that helped us to reduce the dependence on d in our update times.

5. REFERENCES

[1] `http://www.opengeospatial.org/`.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[3] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *FOCS*, pages 363–372, 2011.

[4] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.

[5] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632, 2002.

[6] Michael Benedikt and Leonid Libkin. Exact and approximate aggregation in constraint query languages. In *PODS*, pages 102–113, 1999.

[7] J.L. Bentley. Algorithms for Klee's rectangle problem. Unpublished notes, Computer Science Department, Carnegie Mellon University, 1978.

[8] Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *SODA*, pages 708–713, 2006.

[9] Vladimir Braverman and Rafail Ostrovsky. Recursive sketching for frequency moments. *CoRR*, abs/1011.2571, 2010.

[10] Vladimir Braverman and Rafail Ostrovsky. Zero-one frequency laws. In *STOC*, pages 281–290, 2010.

[11] Mengchu Cai, Dinesh Keshwani, and Peter Z. Revesz. Parametric rectangles: A model for querying and animation of spatiotemporal databases. In *EDBT*, pages 430–444, 2000.

[12] A. Robert Calderbank, Anna C. Gilbert, Kirill Levchenko, S. Muthukrishnan, and Martin Strauss. Improved range-summable random variable construction algorithms. In *SODA*, pages 840–849, 2005.

[13] Timothy M. Chan. A (slightly) faster algorithm for Klee's measure problem. In *SoCG*, pages 94–100, 2008.

[14] Timothy M. Chan and Mihai Patrascu. Counting inversions, offline orthogonal range counting, and related problems. In *SODA*, pages 161–173, 2010.

[15] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *ICALP*, pages 693–703, 2002.

[16] Eric Y. Chen and Timothy M. Chan. Space-efficient algorithms for Klee's measure problem. In *CCCG*, pages 27–30, 2005.

[17] Bogdan S. Chlebus. On the Klee‘s measure problem in small dimensions. In *SOFSEM '98: Proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics*, pages 304–311, London, UK, 1998. Springer-Verlag.

[18] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, page 449, 2004.

[19] Graham Cormode and S. Muthukrishnan. Estimating dominance norms of multiple data streams. In *ESA*, pages 148–160, 2003.

[20] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[21] Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. Approximation techniques for spatial data. In *SIGMOD*, pages 695–706, 2004.

[22] Paul Fischer and Klaus-Uwe HÃűffgen. Computing a maximum axis-aligned rectangle in a convex polygon. *Inf. Process. Lett.*, 51(4):189–193, 1994.

[23] Michael L. Fredman and Bruce Weide. On the complexity of computing the measure of $\cup[a_i, b_i]$. *CACM*, 21(7):540–544, 1978.

[24] Hillel Gazit. New upper bounds in Klee's measure problem. *SIAM Journal on Computing*, 20(6):1034–1045, 1991.

[25] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, pages 79–88, 2001.

[26] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. One-pass wavelet decompositions of data streams. *IEEE Trans. Knowl. Data Eng.*, 15(3):541–554, 2003.

[27] Oded Goldreich. A sample of samplers: A computational perspective on sampling. In *Studies in Complexity and Cryptography*, pages 302–332. 2011.

[28] Ralf Hartmut Güting. An introduction to spatial database systems. *VLDB J.*, 3(4):357–399, 1994.

[29] John Hershberger, Nisheeth Shrivastava, and Subhash Suri. Cluster hull: A technique for summarizing spatial data streams. In *ICDE*, page 138, 2006.

[30] Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 373–380, 2004.

[31] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[32] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC*, pages 202–208, 2005.

[33] V. Klee. Can the measure of $\cup[a_i, b_i]$ be computed in less than $O(n \log n)$ steps? In *American Mathematical Monthly*, volume 84, pages 284–285, 1977.

[34] Gabriel M. Kuper, Leonid Libkin, and Jan Paredaens, editors. *Constraint Databases*. Springer, 2000.

[35] Iosif Lazaridis and Sharad Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD*, pages 401–412, 2001.

[36] K. Levchenko and Y.-K Liu. Counting solutions of polynomial equations, 2005. Manuscript.

[37] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.

[38] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SENSYS*, pages 250–262, 2004.

[39] Mark Overmars. Geometric data structures for computer graphics: an overview. In *Theoretical Foundations of Computer Graphics and CAD*, pages 21–49, 1988.

[40] Mark H. Overmars and Chee-Keng Yap. New upper bounds in Klee's measure problem. *SICOMP*, 20(6):1034–1045, 1991.

[41] Rasmus Pagh. Compressed matrix multiplication. In *ICTS*, pages 442–451, 2012.

[42] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient OLAP operations in spatial data warehouses. In *SSTD*, pages 443–459, 2001.

[43] A. Pavan and Srikanta Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM J. Comput.*, 37(2):359–379, 2007.

[44] Florin Rusu and Alin Dobra. Fast range-summable random variables for efficient aggregate estimation. In *SIGMOD*, pages 193–204, 2006.

[45] Gokarna Sharma, Costas Busch, and Srikanta Tirthapura. A streaming approximation algorithm for Klee's measure problem. *CoRR*, abs/1004.1569, 2010.

[46] Cheng Sheng and Yufei Tao. New results on two-dimensional orthogonal range aggregation in external memory. In *PODS*, pages 129–139, 2011.

[47] He Sun and Chung Keung Poon. Two improved range-efficient algorithms for F_0 estimation. *Theor. Comput. Sci.*, 410(11):1073–1080, 2009.

[48] Yufei Tao and Dimitris Papadias. Range aggregate processing in spatial databases. *IEEE TKDE*, 16(12):1555–1570, 2004.

[49] Nitin Thaper, Sudipto Guha, Piotr Indyk, and Nick Koudas. Dynamic multidimensional histograms. In *SIGMOD*, pages 428–429, 2002.

[50] Jan Vahrenhold. An in-place algorithm for Klee's measure problem in two dimensions. *IPL*, 102(4):169–174, 2007.

[51] Donghui Zhang, Alexander Markowetz, Vassilis J. Tsotras, Dimitrios Gunopulos, and Bernhard Seeger. On computing temporal aggregates with range predicates. *ACM Trans. Database Syst.*, 33(2), 2008.