Summarizing Two-Dimensional Data with Skyline-based Statistical Descriptors

Graham Cormode[1], Flip Korn[1], S. Muthukrishnan[2], and Divesh Srivastava[1]

[1] AT&T Labs — Research {graham, flip, divesh}@research.att.com
[2] Rutgers University muthu@cs.rutgers.edu

Abstract. Much real data consists of more than one dimension, such as financial transactions (eg, price $\times$ volume) and IP network flows (eg, duration $\times$ numBytes), and capture relationships between the variables. For a single dimension, quantiles are intuitive and robust descriptors. Processing and analyzing such data, particularly in data warehouse or data streaming settings, requires similarly robust and informative statistical descriptors that go beyond one-dimension. Applying quantile methods to summarize a multidimensional distribution along only singleton attributes ignores the rich dependence amongst the variables.

In this paper, we present new skyline-based statistical descriptors for capturing the distributions over pairs of dimensions. They generalize the notion of quantiles in the individual dimensions, and also incorporate properties of the joint distribution. We introduce *ϕ-quantours* and *α-radials*, which are skyline points over subsets of the data, and propose *(ϕ, α)-quantiles*, found from the union of these skylines, as statistical descriptors of two-dimensional distributions. We present efficient online algorithms for tracking (ϕ, α)-quantiles on two-dimensional streams using guaranteed small space. We identify the principal properties of the proposed descriptors and perform extensive experiments with synthetic and real IP traffic data to study the efficiency of our proposed algorithms.

1 Introduction

Much of the data in warehouses and streams contains multiple attributes (those attributes typically having skewed distributions), and analysis of such data often calls for summarizing relationships between attributes via robust statistical descriptors. Quantiles (e.g., percentiles), applied to singleton attributes independently, are more descriptive than just the median, which are in turn more robust than the mean. However, quantiles ignore the rich dependencies that can exist between attributes. In particular, it is important to understand the trade-offs between attribute value combinations as well as comparisons between data points exhibiting similar trade-offs.

Example. The American College Board measures academic performance using the SAT standardized test, which historically consisted of both math and verbal sections, and scores for each are sent to prospective colleges. Each section on its own is a narrow indicator of overall scholastic aptitude (e.g., a student with a top score on the math section may have weak verbal skills). Knowing percentiles for sections independently does

not reveal the overall dominance of a student. E.g. if 10% of students scored above 1300 on the math section and 10% scored above 1200 on verbal, it does not imply that 10% of students simultaneously achieved these scores on both sections. Ideally, one would like to summarize the distribution of scores having (roughly) the same sectional percentile ratio. So it will be useful to answer questions such as, *"Which students ranked equivalently as well on the math and verbal sections?"* Figure 1 depicts such scores, labeled "0.5-radial" (explained later), using SAT scores for 2244 colleges.[3] Many schools seek balanced candidates; some (e.g., Caltech) have a preference for mathematical prowess over verbal proficiency. Hence, it is useful to understand the trade-offs between score combinations. Another example question is, *"Which math and verbal score combinations dominate (on both math and verbal scores) 90% of students?"* Figure 1 depicts these scores as the curve labeled "0.9-quantour" on the same data. Using both notions one can obtain the "$(0.9, 0.5)$-quantile": the student dominating 90% of all students and amongst balanced students (along the 0.5-radial); or, equivalently, the balanced student amongst students with 90% dominance (along the 0.9-quantour). □

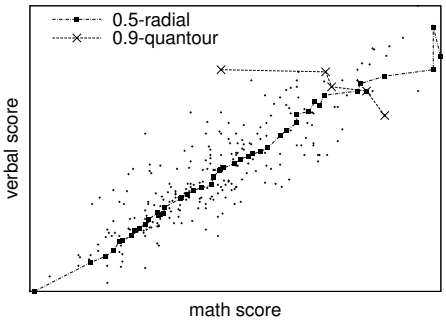


Fig. 1. Radials and quantours on SAT data

Other examples of 2D data analysis arise in the context of flow size distributions within network service providers (where size is measured in both bytes and duration), and financial transactions (price and volume). The outstanding question is how to define and develop such descriptors for multidimensional data. Such a descriptor should reflect the joint distribution while being composed solely of a representative subset of the data points (which can serve as example records). Furthermore, any given input parameter vector should identify a single point in the distribution (though these mappings need not be unique), so that a range of parameter values yields a fixed number of points. Quantiles provide a simple and robust point descriptors of an arbitrary one-dimensional data set (i.e., distribution). But in many problems, such as indexing, going from one dimension to two induces many new difficulties, both conceptual (what are suitable space partitioning techniques in two dimensions?) and technical (how to define the problem declaratively and find a solution efficiently?). To provide an analogous description of multi-dimensional data, several approaches have been considered in the Statistics and Database literature, the three most popular of which are:

– Quantile-quantile (QQ) plots are a well-established tool in data analysis to compare two one-dimensional distributions [11]. Values from the first distribution form the x-axis of the plot, and values from the second form the y-axis. However, this gives a fundamentally one-dimensional view: when applied to the marginals of two-dimensional data, the resulting pairs will likely not be points from the data set. QQ

[3] `http://www.ivywest.com/satscore.htm`

plots allow for the comparison of one-dimensional distributions but are insufficient to give insight into the joint distribution of multidimensional data.

- Tukey proposed an "onion peeling" technique to order points based on the proximity to the "center" of a data set, which is procedurally defined based on recursively stripping away convex hull layers to determine the *contour depth* of a point [22]. However, since an arbitrary number of points may exist at any given depth, points are not uniquely identified by q, so this technique is not a point descriptor. Other approaches, such as multidimensional equidepth histograms [19] have similar deficiencies and are fundamentally ad hoc in nature.
- Lastly, the skyline operator has been proposed to determine the subset of points not dominated by any other points (e.g., "find the skyline of cheap hotels that are close to the beach") [2] and have been generalized to so-called k-skybands [21], that is, points not dominated by more than k points, typically for some small constant k. The skyline (more generally, k-skyband) may contain an arbitrary number of points from the data (perhaps the entire set), and is thus not a point descriptor. Attempts to address this select a subset of the skyline based on orthogonal criteria, rather than distributional properties, such as subspace dominance [4], additional dimensions [12], or the number of distinct points dominated [18].

Our approach is based on computing skylines of subsets of data based on the "depth" of the data within the points in the dataset. This depth is given by removing points which dominate more than a fixed fraction of the whole data set, and so can access points which are far from the traditional skyline or skybands. We are not aware of any prior generalization of skyline queries in this way. This approach is more robust, and functionally rather than procedurally defined (e.g., in comparison with onion peeling).

Our Approach. We introduce two orthogonal notions: the ϕ-*dominance* of a point, which encodes the fraction of the data set dominated; and the α-*skewness* of a point, which is the ratio of its rank in the y-dimension divided by the sum of its ranks in x and y dimensions. We also introduce the notion of ϕ-*quantours* (short for "quantile-contours"), which is a set of points that dominate at most a ϕ-fraction of the multi-dimensional points; and the notion of α-*radials*, which are a set of points having an aspect ratio at most α in their marginal ranks. A point in the data set is uniquely identified by supplying values of ϕ and α. Together, we study the notion of (ϕ, α)-quantiles where points satisfy both α-radial as well as ϕ-quantour properties; thus they *simultaneously* capture the notion of being quantiles in *each* of the dimensions as well as in the joint distribution. They generalize the notion of skylines and provide clearly defined point descriptors in multiple dimensions. Parameters ϕ and α are reminiscent of polar coordinates but applied on order statistics rather than values.

Our Contributions. This paper consists of two parts.

- In the first part of the paper, we introduce ϕ-quantours, α-radials, and (ϕ, α)-quantiles as suitable descriptors of multidimensional distributions and describe a simple algorithm for computing them offline; We analyze these properties and illustrate their use in understanding the "local" structure (Section 2) with respect to the overall joint distribution.

– Motivated by the utility of (ϕ, α)-quantiles to understand a distribution, in the second part of the paper we present small-space algorithms for estimating ϕ-dominance and α-skewness at streaming speeds, with provable guarantees. While tracking (ϕ, α)-quantile points over a stream history may be useful for some applications, in many streaming scenarios only newly-arriving points can be acted upon (e.g., due to time criticality). Thus, the online problem we study is the following. Given a range of ϕ-dominance and α-skewness that are of interest (perhaps obtained by offline distributional analysis described in the first part), the (ϕ, α)-values of incoming points are monitored for matches. For example, if an incoming IP flow record exhibits high α-skew at high ϕ-dominance with respect to respective packet and byte counts, it may be a candidate for more thorough inspection.
  Our algorithms take a stream of points in two dimensions and create compact summaries that can answer such (ϕ, α)-quantile queries accurately. We derive these algorithms by constructing a variety of novel combinations of algorithms for quantiles in one dimension, building on prior work. We consider three fundamental approaches to building these combinations: the cross-product approach, the deferred-merge approach and the eager-merge approach. Each of these has different properties in terms of the space required and the amortized cost per point in the stream. From the viewpoint of real life applications, our methods are able to process more than a hundred thousand flows a second when monitoring IP flows on the stream, and as such are suitable for large Internet Service Provider applications.
– Finally, we perform a detailed experimental study of the online algorithms, using real IP network traffic data as well as synthetic data, to study the space and speed efficiency of these algorithms.

2 Preliminaries

We formally define quantile concepts and problems in two dimensions. We first state the definition of quantiles in one dimension, then show how these can extend to higher dimensions. In one dimension, we consider an input of N items. If we sort the input, and pick out a which is the ith in the sorted order, we say that the rank of a, $\mathrm{rank}(a)$, is i; alternatively, a dominates i points. Given an item a (which may or may not be present in the input), its rank is its position within the sorted input.[4] Throughout we use ϵ to denote the permitted tolerance for error. A *one-dimensional quantile query* is, given ϕ, and an error tolerance ϵ to return a so $(\phi - \epsilon)N \leq \mathrm{rank}(a) \leq (\phi + \epsilon)N$. E.g. finding the median corresponds to querying for the $\phi = \frac{1}{2}$-quantile. We also make use of the stronger error guarantee that is the "biased" quantiles requirement which asks to find a so that $\mathrm{rank}(a) \in (1 \pm \epsilon)\phi N$ [7].

Problem Definition and Discussion. In two dimensions, there is no longer a unique single descriptor of the dominance relationships of the points. The input consists of a stream of items: now these items are points in 2-dimensional space, drawn from a domain of size U, so that each coordinate is in the range $[0 \ldots U-1]$. Let P be the set of N points.

[4] Hence the rank of an item which appears multiple times in the input is a range of positions where the same item occurs in the sorted input.

Definition 1. *The* ϕ-dominance *of a point* $p = (p_x, p_y)$ *is the fraction* ϕ *of points from the input that are* ϕ*-dominated by* p*, i.e.,* $q \in P, (q_x \leq p_x) \wedge (q_y \leq p_y)$*. Let* $\mathrm{rank}(p)$ *be the number of points from the input* $q = (q_x, q_y)$ *such that* p ϕ*-dominates* q*.*[5] *The* ϕ*-dominance of* p*,* $\phi(p)$*, is* $\mathrm{rank}(p)/N$*.*

This notion of dominance is reversed from traditional examples of dominance in skyline computations, but this does not materially affect the definition. From this, we can define a skyline-like operator which identifies points with similar dominance that are not themselves dominated.

Definition 2. *Given a set of points* P*, let* P_ϕ *be the subset of points such that* $P_\phi = \{p \in P | \phi(p) \leq \phi\}$*. Define the* ϕ-dominance quantile contour*, or* ϕ*-quantour for short, as the skyline of* P_ϕ *using the* ϕ*-dominance relation.*

Thus the ϕ-quantour selects those points such that their dominance is at most ϕ, and they are not dominated by any other points with dominance at most ϕ. This definition is carefully chosen so that it is well defined for any $\frac{1}{N} \leq \phi \leq 1$. When $\phi = 1$, this maximal quantour "touches the sky", that is, it is identical to the standard skyline.

Definition 3. *Given a point* p*, we define its* α-skewness *as* $\alpha(p) = \mathrm{rank}_y(p)/(\mathrm{rank}_x(p) + \mathrm{rank}_y(p))$.[6] *Intuitively, this shows how skewed this point is in terms of the ordering of its two dimensions. We similarly set* P_α *to be the subset of points such that* $P_\alpha = \{p \in P | \alpha(p) \leq \alpha\}$*. We say a point* p α-dominates *q if* $p_y > q_y$ *and* $p_x < q_x$ *(and hence* $\alpha(p) > \alpha(q)$*). We define an* α*-radial based on* P_α*, as the skyline of* P_α*, using the* α*-dominance relation.*

Our definition of α means every point has $0 < \alpha < 1$. Intuitively, $\alpha = \frac{1}{2}$ is 'balanced' between x and y dimensions. If we reflect all points in the line $y = x$ to generate a new point set $P^r = \{p^r = (p_y, p_x) | (p_x, p_y) \in P\}$, then the new $\alpha(p^r) = 1 - \alpha(p)$, showing the symmetry of the definition. Our notions of ϕ-dominance and α-dominance are chosen to ensure that, given any two points p and q, either one ϕ-dominates the other, or one α-dominates the other. From studying the definitions, the dominating point will be the one with maximum y value; if they share the same y value, then it is the one with the greater x value. Thus we can combine these two notions and define a unique point estimator.

Definition 4. *We define the* (ϕ, α)*-quantile as follows: we take the skyline of* $P_\phi \cap P_\alpha$ *based on* ϕ*-dominance and take the skyline of the result with* α*-dominance. What remains is defined to be the* (ϕ, α) *quantile.*

Example. Our two-dimensional definitions are illustrated in Figure 2 (points from the same quantours and radials are connected by line segments for illumination). It shows a set of 20 points in two dimensions. For each point we show its (ϕ, α) value:

[5] Technically, the rank of a point can be a range when points from the input share the same coordinates. For simplicity of presentation, we avoid further discussion of this issue and treat rank as if it gives a single value (our results hold when it is a range).

[6] Hence the α-skewness is a range for points having an x- or y-value that appears multiple times in the input; again, we treat α as a unique value for simplicity of presentation.

the point marked p with $(\phi, \alpha) = (0.4, 13/23)$ ϕ-dominates 8 points (including itself), and has $\text{rank}_y = 13$ and $\text{rank}_x = 10$, and so has $\phi = 0.4$ and $\alpha = 13/23$. It falls on the intersection of $\phi = 0.5$ quantour and the $\alpha = \frac{2}{3}$ radial, and therefore is the unique $(0.5, \frac{2}{3})$ quantile.

For the $(0.5, 0.5)$ quantile, there are several possible points that have $(\phi \leq 0.5, \alpha \leq 0.5)$: these are $P_\alpha \cap P_\phi = \{(0.05, 1/2), (0.1, 3/8), (0.1, 2/9), (0.25, 2/5), (0.2, 3/16), (0.35, 2/7), (0.35, 10/21), (0.45, 9/26)\}$. However, the unique point q at $(0.35, 10/21)$ either ϕ-dominates or α-dominates every other point in $P_\alpha \cap P_\phi$, and is therefore *the* $(0.5, 0.5)$ quantile. As noted next, this is the point with the greatest y rank amongst the set which obey the (ϕ, α) predicates, and it falls on the $\alpha = 0.5$-radial.

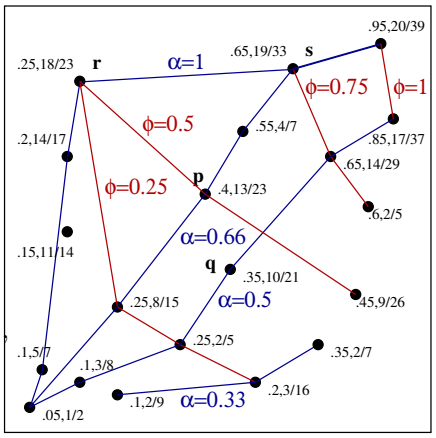


Fig. 2. Sample data set with (ϕ, α) values for each point shown along with ϕ-quantours and α-radials for $\phi = \frac{1}{4}, \frac{1}{2}, \frac{3}{4}$ and $\alpha = \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, 1$.

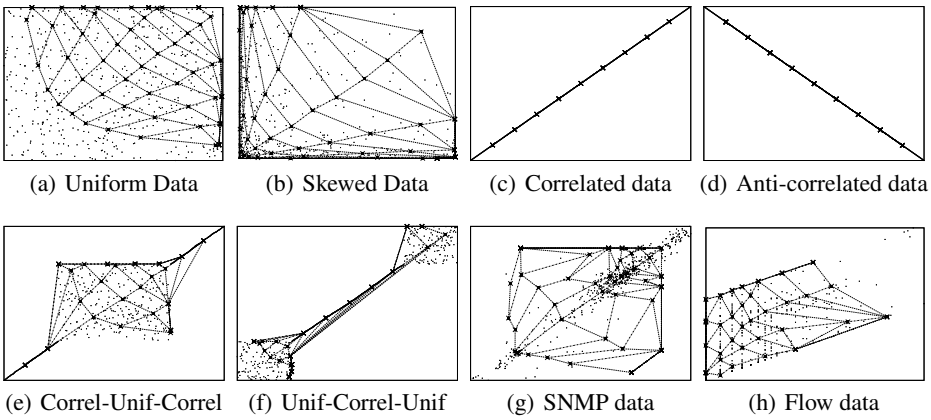
□

Properties of (ϕ, α)-quantiles, ϕ-quantours and α-radials. One can readily verify the following statements:

1. A unique point from the input is found. This follows since, after taking the ϕ-dominance skyline, we obtain a set of points such that no pair is comparable under ϕ-dominance. Thus, they must all be comparable under α-dominance, and hence there is a unique maximal point.
2. The order of the taking the ϕ-dominance skyline and the α-dominance skyline is unimportant. Moreover, the unique point that is returned is the point in $P_\phi \cap P_\alpha$ which has the greatest y value and, if more than one has this y value, the one amongst them with the greatest x value.
3. The returned point lies on the α-radial or the ϕ-quantour, or possibly both if they intersect. When the maximal points on the α-radial and ϕ-quantour have differing y-values, then the (ϕ, α)-quantile is guaranteed to be on the α-radial.
4. For any two input points p, q, we have $\phi(p) = \phi(q) \wedge \alpha(p) = \alpha(q) \Leftrightarrow p = q$. In other words, all distinct input points have distinct (ϕ, α) values. This is seen by observing that if two points share the same α-value then one must ϕ-dominate the other. So p is *the* $(\phi(p), \alpha(p))$-quantile.

Thus we have a robust definition which selects a unique point p from the input having $\alpha(p) \leq \alpha$ and $\phi(p) \leq \phi$.

Exact Algorithm. We note that one can compute (ϕ, α) quantiles with similar time cost to computing quantiles in one dimension. We omit the details for brevity; the main idea is to scan the data in an appropriate order, and track certain information to rapidly compute the ϕ-dominance of a point. The result we claim is that given $O(N \log N)$ preprocessing, we can find the (ϕ, α)-quantiles (in two-dimensions) in time $O(N)$ per query.



(a) Uniform Data (b) Skewed Data (c) Correlated data (d) Anti-correlated data

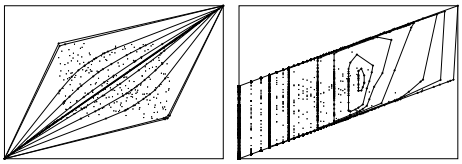
(e) Correl-Unif-Correl (f) Unif-Correl-Unif (g) SNMP data (h) Flow data

Fig. 3. Radials and quantours on synthetic and real data

Other Dimensionalities. Note that in one-dimension, where the notion of α does not apply, this definition naturally collapses back to the familiar definition of ϕ-quantiles. We can also generalize these definitions to higher dimensions: the same notion of ϕ and ϕ-dominance translate immediately. One can also define $d-1$ new α-dominances between the d dimensions: let $\mathrm{rank}_i(p)$ denote the rank of the projection of p on the ith dimension of d dimensions. We define $\alpha_i(p) = \mathrm{rank}_i(p)/\sum_{j=1}^{d} \mathrm{rank}_j(p)$. Thus, we have $\sum_{i=1}^{d} \alpha_i(p) = 1$, and in two dimensions we recover our original definition. Since our focus is primarily on the two-dimensional case we will only briefly mention extensions to higher dimensions later.

Nature of ϕ-Quantours and α-Radials Figure 3 plots the ϕ-quantours, α-radials and (ϕ, α) quantiles of several synthetic data sets, for $\phi \in \{0.1, ..., 0.9\}$ and $\alpha \in \{0.1, ..., 0.9\}$. Comparing Figure 3(b) with Figure 3(a) demonstrates how skew affects the angles between radials, causing them to diverge when both x and y have higher skew. In Figure 3(c), every point p has $\alpha(p) = 0.5$, effectively collapsing to 1D quantiles. In Figure 3(d), every point p has $\phi(p) = \frac{1}{N}$ (i.e., every point is on the skyline), effectively collapsing to 1D quantiles along the skyline. Figures 3(e) and 3(f) demonstrate how (ϕ, α) quantiles follow the "shape" of a point cloud on hybrid data sets having (and lacking) regional correlations. In Figure 3(f), the reason why only points above the diagonal merge into the diagonal is due to the definition forcing α-radial points p to have $\alpha(p) \leq \alpha$.

Figures 3(g) and 3(h) give examples of quantours and radials of real data sets, plotted in log-log scales. In Figure 3(g), which plots outbound versus inbound traffic volumes, the correlation varies by region of the plot, as indicated by how the radials "bend inwards": at low values there is high correlation (acute quantour angles), at medium values low correla-



(a) Correl-Unif-Correl (b) Flow data

Fig. 4. Depth contours on synthetic & real data

tion (obtuse angles), and at high values again high correlation (acute angles). Compare this with Figures 3(a) and 3(b), where there is no correlation between x and y. Hence, inbound and outbound traffic is balanced for medium traffic levels, but the traffic parity doesn't remain when the throughput in either direction is too large or small. In Figure 3(h), which plots flow sizes in bytes versus packets, the distributions of both attributes are skewed, as indicated by the divergence of the radials away from the center. The curvature of the 0.5-radial shows that the relationship between packets and bytes is different for small "balanced" flows than larger ones, perhaps indicating distinct application types. For contrast, Figure 4 shows the data from Figures 3(e) and 3(h) plotted with depth contours. We argue that these plots are less informative, and more idiosyncratic, than the quantour/radial plots: Figure 4 is unable to capture the local information as well as Figure 3(e) and 3(h). Quantile-quantile plots are also unsuitable for these examples: Figures 3(a), 3(c), 3(e) and 3(f), all have the same set of x and y values, and so would be indistinguishable in a QQ-plot.

3 Streaming algorithms

We now define and solve approximate versions of the problem which will allow us to reduce the amount of space required to answer them. While in one dimension it is possible to give relative error $(1 \pm \epsilon)$ estimates of ϕ-dominance, the same is not true in higher dimensions (we omit the formal information theoretic proof for brevity). Instead, we formalize the requirements as:

Definition 5. *Given a stream of data points in two dimensions, the* approximate (ϕ, α) quantile problem *is to process the stream so that, given any point p, we return an approximation $(\hat{\phi}(p), \hat{\alpha}(p))$ satisfying:*

$$\phi(p) - \epsilon \leq \hat{\phi}(p) \leq \phi(p) + \epsilon \text{ and } (1-\epsilon)\alpha(p) \leq \hat{\alpha}(p) \leq (1+\epsilon)\alpha(p).$$

This allows the accurate estimation of ϕ and α values for any point, whether in the input data, or not (e.g. answering "what if" queries, such as how a particular flow would rank amongst the recently observed data). We can draw sample points from the input as quantile points, since our algorithms keep track of a representative set of input points.

3.1 Algorithmic Approach

We provide a selection of algorithms to solve the above approximate problem over streaming data. We separate the two key components of the approximate (ϕ, α) quantile problem: estimating α values and estimating ϕ values. Estimating α is relatively straightforward, since it can be found by combining independent estimations of one dimensional quantiles with appropriate guarantees. Thus, the bulk of our challenge comes in estimating ϕ-dominance of points. Our approach uses techniques from the one-dimensional problem to summarize the two-dimensional data by dividing the data on each axis. We propose three classes of algorithms for combining one-dimensional summaries in order to compute (ϕ, α)-quantiles. These include some algorithms previously proposed for the related but distinct problem of computing rank queries in two-dimensions. Here, we provide a more general framework and demonstrate the different combinations that are possible to combine to form other examples of such classes.

The two-dimensional algorithms retain a subset of points from the input, and additional information to allow the estimation of ϕ and α values of points. We primarily

Method	Space cost	Update time	Compressing	Mergability	Hierarchical
GK [13]	$O(\frac{1}{\epsilon}\log \epsilon N)$	$O(\log(\frac{1}{\epsilon}\log \epsilon N))$	Yes	Weakly	No
QD [25]	$O(\frac{1}{\epsilon}\log U)$	$O(\log\log U)$	Yes	Strongly	Yes
BQ [7]	$O(\frac{1}{\epsilon}\log U \log \epsilon N)$	$O(\log\log U)$	No	Strongly	Yes

Fig. 5. Properties of one-dimensional quantile summary algorithms.

consider algorithms which support several operations: INSERT(x), which takes a new value $x \in [0 \ldots U-1]$ and updates the summary accordingly; $\hat{\phi}(q)$ which returns the approximate ϕ-dominance of point q; $\hat{\alpha}(q)$ which returns the approximate α-dominance of point q; and COMPRESS, which compacts the data structure.

3.2 Properties of one-dimensional algorithms

We first summarize some of the properties of one-dimensional quantile summary algorithms that we use as the building blocks of our two-dimensional algorithms—note that not all algorithms have all properties, which affects which combinations are possible.

- Mergable: an algorithm is considered (strongly) *mergable* if two summaries of different inputs can be combined to create a summary of the union of the two inputs. Our focus is on summaries that can be merged arbitrarily many times and still retain the same asymptotic space bounds. Other summaries are weakly mergable, in that their output can be merged to answer a query on the union of their inputs, but there is no guarantee that the size of the merged summary is less than the sum of sizes of the original summaries.
- Compressing: a quantile algorithm is *compressing* if it stores items from the input and, when the size of the summary is being reduced, tuples are compressed together by summing the counts of particular items or ranges from the input.
- Hierarchical: a quantile algorithm is *hierarchical* if it follows a pre-determined, hierarchical approach to merging: a tree-structure is placed over the domain, and merges only occur between child nodes and their parent.

We briefly outline existing "sample-based" algorithms (which maintain summaries based on selecting points from the stream):

- The Greenwald-Khanna algorithm (GK) [13] retains a set of tuples consisting of an item from the input, a count of how many items have been merged into that tuple, and an upper bound on the rank of the item.
- The Quantile Digest algorithm (QD) [25] retains a set of tuples, where each tuple consists of an item or (dyadic) range of items from the input and a count of how many items have been merged into that tuple.
- Biased Quantiles (BQ) [7] uses Quantile Digest-like data structure with different manipulation routines to estimate the dominance of a point with stronger relative error guarantees, with slightly higher space usage.

QD and BQ do not by default retain points from the input. But it is straightforward to augment them to do so: for every tuple that relates to a range of items, we additionally keep some point from the input that fell in that range, and merge these appropriately. Figure 5 lists the key properties of these algorithms. Randomized methods, such as random sampling, can give similar guarantees, but our focus is on stronger deterministic

guarantees, so we do not discuss these. Each of the algorithms we do consider allows us to 1DINSERT a new item, 1DCOMPRESS the structure to compact the size to its theoretical bounds, and 1DQUERY to find the approximate rank of a point.

Using 1D algorithms to approximate α. Computing $\alpha(p)$ can be carried out with relative error in small space:

Theorem 1. *Using space $O(\frac{1}{\epsilon}\log U \log \epsilon N)$, we can take any point q and compute an approximation $\hat{\alpha}(q)$ such that $(1-\epsilon)\alpha(q) \leq \hat{\alpha}(q) \leq (1+\epsilon)\alpha(q)$.*

Proof. Recall that $\alpha(q) = \mathrm{rank}_y(q)/(\mathrm{rank}_x(q) + \mathrm{rank}_y(q))$. By maintaining a biased quantile data summary (BQ) on the x-values and y-values of points independently, we can approximate rank_x and rank_y with $\hat{r}_x$ and $\hat{r}_y$ such that $(1-\epsilon)\,\mathrm{rank}_x(q) \leq \hat{r}_x(q) \leq (1+\epsilon)\,\mathrm{rank}_x(q)$ and $(1-\epsilon)\,\mathrm{rank}_y(q) \leq \hat{r}_y(q) \leq (1+\epsilon)\,\mathrm{rank}_y(q)$. Thus, finding $\hat{\alpha}(q) = \hat{r}_y(q)/(\hat{r}_x(q) + \hat{r}_y(q))$ ensures that the relative error is between $\frac{1-\epsilon}{1+\epsilon} \geq 1-2\epsilon$, and $\frac{1+\epsilon}{1-\epsilon} \leq 1+3\epsilon$, for $\epsilon \leq \frac{1}{3}$. We rescale ϵ by a factor 3, which does not affect the asymptotic space costs of the BQ summary, whose space bounds follow from [7].

The next three sections outline three classes of algorithms, and for each give sample instantiations based on combining appropriate one-dimensional algorithms from the list above. For reasons of brevity, we do not give complete proofs of all properties of the algorithms in this presentation, but instead provide an outline of why they hold. The main properties of each algorithm are summarized in Figure 6.

3.3 Cross-Product Algorithms

Our first approach to tracking quantile information in multiple dimensions is the cross-product approach, based on keeping one-dimensional compressing sample-based summaries on each dimension independently.

Update Processing in Cross-Product. Each 1D summary consists of a set of items and ranges from each dimension, and we maintain information about the cross-product across dimensions: if we keep information on a set X of points or ranges from the x-dimension, and a set Y on the y-dimension, then we will maintain information on the Cartesian product $X \times Y$. For each cell in this cross-product, we maintain a count of the number of input items associated with the cell, and a point from the input that falls in the cell, if the count is non-zero. The counts must satisfy the property that summing the counts of all cells within a rectangle gives a lower bound on the number of input points within that rectangle.

Periodically, a 1DCOMPRESS operation is run on first X and then Y. When the 1D algorithm merges two tuples together, the corresponding cells from $X \times Y$ are also merged. The count of the merged cell is the sum of the counts of the cells which went to form it. The retained point is chosen arbitrarily from the points of the merged cells. When INSERT is run on a new point $p = (p_x, p_y)$, we update $X \times Y$ accordingly to reflect the changes from the insertion to X and Y. Typically, this means inserting p_x into X and p_y into Y, so we update $X \times Y$ with $\{p_x\} \times Y$, $X \times \{p_y\}$ and (p_x, p_y). We set the count $(p_x, p_y) = 1$ (if (p_x, p_y) already exists in $X \times Y$, we increment its count) and the count of all other new cells to zero (if they already exist, we leave them unchanged).

Estimation and Accuracy on Cross-Product. In order to compute $\hat{\phi}(q)$, we compute the count of all cells dominated by q. The approximation error in our response comes because q may fall within a cell: all cells below and to the left of q contain points that are strictly dominated by q, and all cells above or to the right contain points that are not dominated by q; this leaves the cells containing the x-value of q, and the cells containing the y-value of q. The properties of the one-dimensional structures ensure that this uncertainty is limited to ϵN. The space used by this algorithm depends on the product of the sizes of the one-dimensional summary structures used. Various combinations are possible: GK×GK, QD×QD (which was considered in [26]) or even GK×QD.

Instantiation: Cross-product with GK×GK . Since the GK algorithm typically attains the best space usage on one-dimensional data, it is expected that GK×GK will attain the best space usage of the cross-product algorithms. The space bounds follow from [26], and restated here along with our improved time bounds. Asymptotically, the space usage of GK×GK is bounded by $O(\frac{1}{\epsilon^2}\log^2(\epsilon N))$. The time cost of all cross-product algorithms can be somewhat high. For efficiency, rather than explicitly materializing all cells, our idea is to use a hash table to store only those cells (x, y) in the grid that have a non-zero count. GK (along with other one-dimensional algorithms) adds a new item to the data structure for each INSERT operation. In two dimensions this adds $O(\frac{1}{\epsilon}\log(\epsilon N))$ new cells, but by using the hash table approach, we only have to do $O(1)$ operations since only a single new cell is created with a non-zero count. COMPRESS requires time linear in the (worst case) size of the data structure, $O(\frac{1}{\epsilon^2}\log^2(\epsilon N))$: it consists of compressing each of the one-dimensional data structures independently, and when they merge two of their tuples, merging together the cells associated with those rows/columns. Merging a row takes time linear in its size, and compressing each 1D data structure also takes linear time, so the total time cost of COMPRESS is worst case bounded by the size of the data structure. By performing COMPRESS after every $O(\frac{1}{\epsilon}\log(\epsilon N))$ INSERT operations, the space bounds are preserved, while the amortized time cost is $O(\frac{1}{\epsilon}\log(\epsilon N))$ per update. The time to compute $\hat{\phi}(p)$ is at most linear in the size of the data structure, and to compute the $\hat{\phi}$ of every stored point at most logarithmically longer in the size of the data structure, using the exact algorithm of Section 2.

3.4 Deferred-Merge Algorithms

Our next approach fixes an ordering on the dimensions, and runs a compressing algorithm on the primary (x) dimension with uniform guarantee ϵ_x. It runs multiple instances of (strongly) mergable algorithms on the secondary (y) dimension with parameter ϵ_y. Dominance queries require the merging of multiple secondary data structures to answer, but we defer this merging to query time, hence "deferred-merge". So for the primary axis, we can use either GK or QD; for the secondary axis, we can use QD because of its mergable properties.

Update Processing in Deferred-Merge. For each input point p, we first 1DINSERT p_x into the compressing algorithm on the x-dimension. Instead of just keeping a count of the number of items retained in each tuple, we also keep a second one-dimensional quantile data structure that summarizes the y-values of all points that are summarized in the tuple. So after finding the data structure tuple to insert p_x into, we insert p_y into its associated summary. For compression of the data structure, we first run 1DCOMPRESS

on the one-dimensional summary of x-values. If we merge tuples in this structure, then we also merge together their summaries of y-values and 1DCOMPRESS the result.

Estimation and Accuracy on Deferred-Merge. In order to approximate $\phi(q)$, we can query q_x to find the summary containing q_x, and then merge together all summaries of y-values to the left of this, and query the resulting merge structure with q_y. The approximation error from this approach comes from two sources: uncertainty due to the querying on x-values, and uncertainty on y-values. The guarantees of the summaries on x-values ensure that the uncertainty is at most $\epsilon_x N$ points; similarly, posing a query to the merged summary of y-values gives a guarantee depending on ϵ_y. In order to get the required accuracy bounds, we set the parameters ϵ_x and ϵ_y less than or equal to $\frac{\epsilon}{2}$, giving accuracy ϵN or better for each query. Consequently, this algorithm solves the approximate (ϕ, α) quantiles problem. Lastly, we observe that the space bound of merge-based algorithms is at most the product of the space bounds of the algorithms on each axis. We can instantiate the deferred merge algorithms with either GK or QD on the primary (x) axis, but require a strongly mergable algorithm for the secondary (y) axis, which allows us to use GK×QD or QD×QD.

Instantiation: Deferred-merge algorithm with GK×QD. GK is a compressing algorithm that typically achieves the best space in 1D; QD is a mergable algorithm that also has relatively small space cost. The worst case bound on the space needed is the product of the space bounds: $O(\frac{1}{\epsilon^2} \log(\epsilon N) \log U)$. To INSERT a new point, we first insert the p_x into the GK structure, and store p_y along with the inserted points itself as a QD summary of size 1. To COMPRESS the summary, we run the one-dimensional GK-COMPRESS on the GK structure with error parameter $\epsilon_x = \epsilon/2$, and when two tuples in GK are merged, we also merge their corresponding QD summaries (and then run the one-dimensional QD-COMPRESS on the result with error parameter $\epsilon_y = \epsilon/2$). The time to perform the compression is thus worst case bounded by time linear in the total data structure. This can be amortized by running COMPRESS only after every $O(\frac{1}{\epsilon^2} \log(\epsilon N) \log U)$ updates, which retains the asymptotic space bounds, and ensures that the update cost is dominated by the cost of inserting into one GK structure and one QD structure, which is $O(\log(\frac{1}{\epsilon}) + \log\log U)$. The overall uncertainty in ϕ-dominance queries is at most ϵ.

An important feature of the merge based algorithms is that the size of the data structure is never more than the size of the input, since each input point corresponds to at most one tuple in the summary. This is in contrast to the next methods we consider, which have the potential to represent each point multiple times during the early phases of the algorithm.

3.5 Eager-Merge Algorithms

The class of eager-merge algorithms also combine one-dimensional algorithms. They use a compressing hierarchical algorithm on the primary (x) axis, with uniform guarantee ϵ_x. Rather than demanding that the algorithm on the secondary (y) axis be strongly mergable (as in the deferred-merge case), they eagerly compute the results of merging by inserting the y-dimension of each input point into a summary at each level of the hierarchy, and (weakly) merge appropriate outputs from these structures at query time.

The first such algorithm was proposed in [14]. Here, we generalize the description, and give details for completeness.

Update Processing In Eager-Merge. For each materialized node in the primary data structure, we maintain a second data structure on the y-dimensional values of all points allocated to this node *or any of its descendants*, with an accuracy guarantee set to ensure accurate answers. To perform an INSERT operation on a new point, we first find the node in the x-structure corresponding to p_x from the inserted point. We then 1DINSERT p_y into the corresponding y-summary of that node, and also into the y-summaries of every ancestor of the node. When we create a new node based on an input point p, we store p along with that node. COMPRESS takes place firstly over the data structure on the x-axis, and then on each of the data structures covering the y-axis contained within it. We run the 1DCOMPRESS routine over the x structure and when nodes are merged, only their associated counts are updated. The y-summaries corresponding to the deleted children can simply be deleted: they are not merged into their parent, since during insertions, the result of the merge is already (eagerly) computed, by ensuring that every inserted point was put into the y-summaries of every ancestor. After deleting y-summaries, we then perform a COMPRESS on those that remain.

Estimation and Accuracy in Eager-Merge. The product of the x and y space bounds gives a naive space bound, but tighter bounds are obtained using the fact that each point is represented at most once per level. We need to use a hierarchical algorithm such as QD on the primary (x) axis, and a (weakly) mergable algorithm on the secondary axis.

Instantiation: Eager-merge algorithm with QD×QD. Our eager-merge algorithm uses QD as the method on the first dimension, since this method is hierarchical, and also uses QD on the second dimension. (An alternative is QD×GK, but QD×QD yields better worst case space bounds.) Rather than using the number of points within the y-summary as the basis of the threshold for compressing, we use a threshold based on N, the total number of points in the data structure (the same idea was used in [14], though we obtain different bounds due to implementation choices). This is because the overall error guarantee is in terms of ϵN, and to give a tight space bound. Within each y-summary, we set a local error tolerance of $\frac{\epsilon}{2 \log U}$. This is chosen to ensure that when these are summed over $\log U$ different summaries, the total error will be bounded by $\epsilon/2$.

INSERT operations take time $O(\log U \log \log U)$: we perform 1DINSERT operations for $O(\log U)$ nodes in the x-dimensional summary, and each of those can be completed in time $O(\log \log U)$ on the y-dimensional summaries [7]. A COMPRESS operation takes time linear in the size of the data structure, since it reduces to running 1DCOMPRESS on multiple one-dimensional data structures, each of which takes time linear in the size of their substructure. The amortized update time is therefore $O(\log U \log \log U)$.

To answer a ϕ-dominance query given q, we find a set of nodes in the x structure to query, by representing q_x as a union of disjoint ranges from the hierarchy. The binary tree structure of the QD algorithm ensures that there are at most $\log U$ nodes in this set. For each node in the set, we query its corresponding y-summary with q_y and sum the outputs to obtain the estimate of $\mathrm{rank}(q)$: q_x is broken into two y-summaries, one

Type	Instance	Space	Amortized time	$\hat{\phi}$-query time
Cross-product	GK×GK	$O(\frac{\log^2 \epsilon N}{\epsilon^2})$	$O(\frac{\log \epsilon N}{\epsilon})$	$O(\frac{\log^2 \epsilon N}{\epsilon^2})$
Deferred-merge	GK ×QD	$O(\frac{\log U \log \epsilon N}{\epsilon^2})$	$O(\log \frac{\log U}{\epsilon})$	$O(\frac{\log U \log \epsilon N}{\epsilon^2})$
Eager-merge	QD×QD	$O(\frac{\log^3 U}{\epsilon})$	$O(\log U \log \log U)$	$O(\frac{\log^3 U}{\epsilon})$

Fig. 6. Comparison of bounds for different instantiations of two-dimensional data structures

on the first half of the horizontal span, another on the next quarter. We get an accurate count of the number of points within the queried region: summing the accuracy bounds over the at most $\log U$ queries gives error at most $\epsilon/2$. The uncertainty due to the query on the x-axis is also at most $\epsilon/2$, from the accuracy bound on the x-data structure, so the total error is at most ϵ. Since queries probe at most $\log U$ y-summaries, each in time $O(\log U)$, the total time cost is $O(\log^2 U)$ per query after a COMPRESS has updated counts in time $O(\frac{\log^3 U}{\epsilon})$.

3.6 Comparison

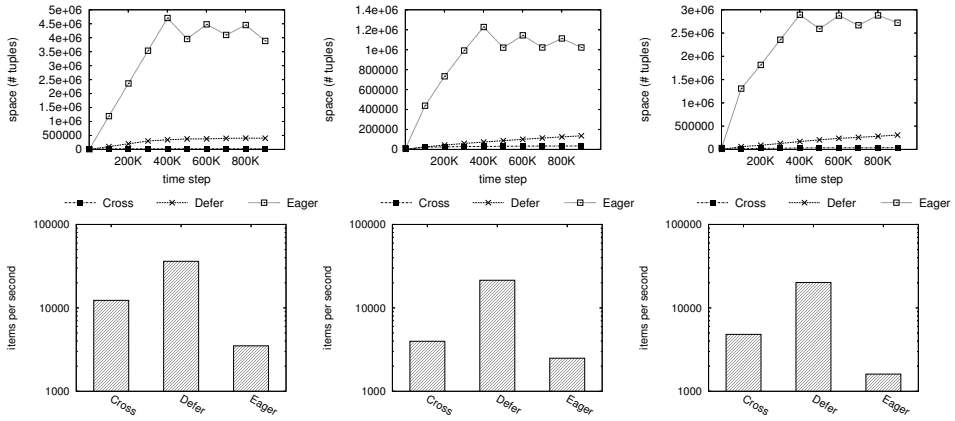
We summarize the space and time bounds of various instantiations of the three different approaches in Figure 6. Initially, it is hard to compare them, since the relative asymptotic cost depends on the setting of parameter ϵ, relative to $\log U$. Comparing eager-merge costs to deferred-merge, the space cost trades off roughly a factor $O(\frac{1}{\epsilon})$ for one of $O(\log U)$. This suggests that for very fine accuracy situations $\epsilon \ll \frac{1}{\log U}$, the eager-merge approach will win out. Comparing cross-product to deferred-merge, it seems possible that cross-product methods will use less space, especially since GK has been observed to use closer to $O(\frac{1}{\epsilon})$ space in practice. But the amortized running time of the deferred-merge algorithms are exponentially smaller than those of the cross-product algorithms, which can make a big difference over high speed data streams.

In terms of query time for estimating ϕ-dominance, by default all methods require a linear pass over the whole data structure to answer the query, so those with smaller space have faster queries. But, if many queries are being computed in a batch, then the time cost of eager-merge methods can be improved to $O(\log^2 U)$ by taking advantage of the hierarchical structure of the summary to answer queries much faster. This relies on utilizing stored counts within the data structure that are needed by the INSERT and COMPRESS routines. By recomputing these counts in a linear pass before each computation of $\hat{\phi}(p)$, the running time is much reduced for each query.

4 Experimental Results

In this section, we summarize experiments comparing the three classes of algorithms in Section 3 for answering approximate (ϕ, α) estimation queries: cross-product based on GK×GK; deferred-merge based on GK×QD; and eager-merge based on QD×QD.

In all experiments, we run all three instances with the same accuracy requirements ϵ and observe how their space and time bounds vary while they provide the same accuracy guarantee. We report space usage in terms of the number of tuples and as a function of the number of stream tuples that have arrived; we report performance in terms of the *throughput*, that is, the number of tuples processed per second. These experiments were run on a Linux machine with a 2.8 GHz Pentium CPU, 2 GB RAM and 512K cache.



(a) Uniform data (b) Zipfian data (c) Network packets-bytes data

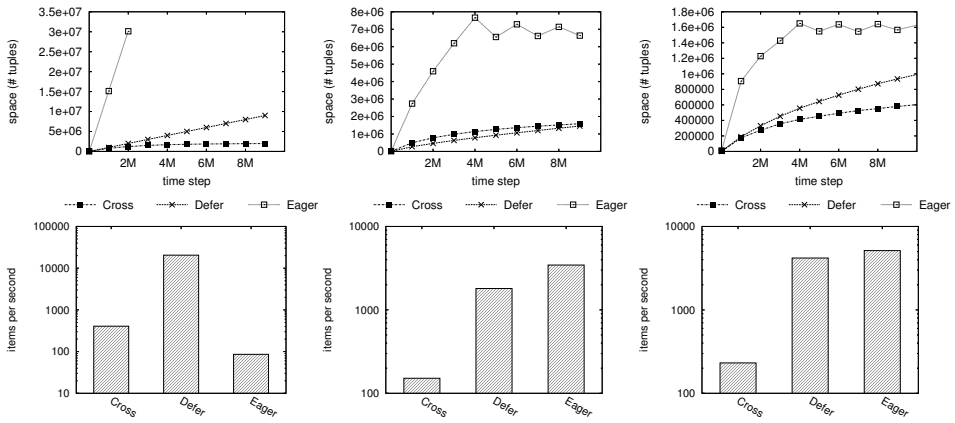
Fig. 7. Space usage and throughput ($\epsilon = 0.01$).

For synthetic data, we generated random uniform data with universe size 2^{32} on each axis independently; we did the same with Zipfian data (skew parameter=1.3); and we generated bivariate Gaussian data sets with varying correlation strengths. We also used real network flow data from a router used in the AT&T Common Backbone and obtained 2D data sets by projecting on the fields (`numPkts`, `numBytes`), (`duration`, `numBytes`), and (`srcIP`, `destIP`).

In addition, we conducted experiments using a live stream of IP packet traffic monitored at a local network interface. The traffic speeds varied throughout the day, with a rate of about 200K TCP packets per second at mid-day. The stream data was monitored using Gigascope, a highly optimized system for monitoring very high speed data streams [8]. Our copy was installed on a FreeBSD machine with a dual Intel Xeon 3.2 GHz CPU and 3.75 GB RAM. The methods were implemented as User-Defined Aggregate Functions (UDAFs) in Gigascope, as explained in [6].

Results on Real and Synthetic Data Sets. Figure 7 shows a space usage and throughput comparison (in log scale) of the different algorithms run with $\epsilon = 0.01$; Figure 8 shows the same with $\epsilon = 0.001$. The algorithms were run longer (up to 10M tuples) in Figure 8 than in Figure 7 (1M tuples), long enough so that the space usage curves can be seen to "level off" and converge to approximately stable values. None of the algorithms was dominant in all cases. Notice that space usage and performance were not always correlated: some cases exhibited a space-time trade-off.

Data skew affected space usage and performance: in general, there was smaller space and faster runtime with increasing skew. Whereas the GK algorithm (in 1D) is impervious to the data values since it only cares about rank-ordering, the universe-based algorithms (eg, QD) benefit from non-uniformity. This can be seen from the space usages in Figures 7(a) and 7(b): the gap between the curves for eager-merge (based on QD) and cross-product (based on GK) narrows significantly. Indeed, real data is often skewed, as is the case with flow data (thought not quite as skewed as the Zipfian data



(a) Uniform data (b) Zipfian data (c) Network packets-bytes data

Fig. 8. Space usage and throughput ($\epsilon = 0.001$).

set). Hence, the space usage on this data set in Figure 7(c) was much more similar to that on Zipfian data in Figure 7(b) than uniform data in Figure 7(a). The deferred-merge algorithm, a hybrid of GK and QD approaches, is asymmetric: it is more efficient when the attribute with higher skew is in the y-axis. We observed up to a factor of 5 difference in the space used by deferred-merge by swapping the ordering of the axes.

Another relevant factor is correlation between attributes, as observed using bivariate Gaussian data of differing covariances. This benefits the cross-product approach because, with increasing correlation, cross-product effectively becomes one-dimensional; correlation did not have much impact on the other classes. Real data, such as flow packets and bytes, often exhibit correlations. Skew, which does not benefit cross-product but benefits the other algorithms, appeared to be more significant in our experiments.

In the streaming scenario, throughput often trumps space efficiency.[7] Therefore, deferred-merge is the overall "safest" algorithm to use. In our experiments, it always had higher throughput than cross-product (often 1-2 orders of magnitude) while being competitive with respect to space usage; and it had as good or better performance than eager-merge, with significantly better space usage. In some cases (e.g., Figure 8(a)), the space usage of eager-merge grew so large that it exceeded the RAM size, causing the system to thrash and resulting in abysmal throughput.

Live Packet Streams. We issued a long-running query to the Gigascope system to maintain per-minute (ϕ, α)-quantiles with $\epsilon = 0.01$, on our methods, over a cumulative window of pairs of approximate flow aggregates on (numPkts, numBytes) grouped by flow. We compared to a 'null' aggregate that just counts updates, which required 85% CPU utilization. Cross-product and eager-merge could not keep up with this stream, but the deferred-merge UDAF achieved processing rates similar to the 'null' aggregate (130-140K per minute) and CPU utilization around 88%.

[7] For example, the Gigascope high-level query processor can make use of as much RAM as available.

5 Related Work

In Statistics, there has been significant work on multidimensional quantile descriptors. A good overview can be found in [24], with some specific approaches in [9, 5, 16]. However, these approaches do not yield point descriptors but algebraic curves. Quantile-quantile (QQ) plots compare the 1D quantiles from each marginal as 2D points. Such plots allow us to compare one-dimensional distributions, but are insufficient to give us full insight into joint distribution of multi-dimensional data.

Similarly, in Computational Geometry, notions of median and other quantiles are procedurally defined in terms of "depths" of point-sets and produce quantile regions [10, 22, 23, 15]. While this may be satisfying for visualization, such region-based quantile descriptors are not point descriptors. Other descriptors such as multidimensional equidepth histograms and one dimensional quantiles on linearized multidimensional data are ad hoc, and have similar deficiencies.

In recent years there has been significant interest in the area of data streams, where the space available for processing is considerably smaller than the input, which is presented in a "one-pass" fashion [1, 20]. As previously noted, there is a wealth of algorithms devoted to the problem of tracking (1D) quantiles in data streams [13, 25, 7]. For multidimensional data streams, prior work has been scant; it is primarily focused on summaries such as histograms [3, 27].

Our three classes of algorithms for combining one-dimensional summaries in order to compute (ϕ, α)-quantiles capture some algorithms previously proposed for the related but distinct problem of computing rank queries in two-dimensions. Algorithms proposed in [26, 14] can be thought of as fitting into our class of "cross-product" and "eager-merging" summaries, respectively. Here, we provide a more general framework and demonstrate the different combinations that are possible to combine to form other examples of such classes. We are not aware of any prior examples which demonstrate the deferred-merging approach that we have proposed. Interestingly, it appears that this class is often the best suited for keeping pace with high streaming data rates.

6 Conclusions

Data in warehouses and streams, such as IP traffic data, are typically multidimensional, and capture relationships between multiple variables. In a variety of applications, one needs simple, statistical point descriptors of such streams. Existing methods use quantiles in single dimensions and therefore miss joint distributional behavior, or give procedural or ad hoc definitions. In this paper, we propose skyline-based statistical descriptors, and introduce ϕ-quantours, α-radials and, in particular (ϕ, α)-*quantiles*. We present fast and small-space streaming algorithms for computing them approximately with guaranteed accuracy by judicious combinations of previously known one-dimensional algorithms. We demonstrate experimentally the efficiency of computing 2D quantiles in data streams with synthetic and real data.

References

1. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *ACM PODS*, pages 1–16, 2002.
2. S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *IEEE ICDE*, pages 421–430, 2001.

3. N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: a multidimensional workload-aware histogram. In *ACM SIGMOD* , 2001.
4. C. Y. Chan, H. V. Jagadish, K-L Tan, A. K. H. Tung, Z. Zhang. On High Dimensional Skylines. In *EDBT*, 2006.
5. P. Chaudhuri. On a geometric notion of quantiles for multivariate data. *Journal of the American Statistical Association*, 91:862–872, 1996.
6. G. Cormode, F. Korn, S. Muthukrishnan, T. Johnson, O. Spatscheck, and D. Srivastava. Holistic UDAFs at streaming speeds. In *ACM SIGMOD* , pages 35–46, 2004.
7. G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In *ACM PODS*, 2006.
8. C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *ACM SIGMOD* , pages 647–651, 2003.
9. J. Einmal and D. Mason. Generalized quantile processes. *Annals of Statistics*, 20(2):1062–1078, 1992.
10. D. Eppstein. Single point estimators, 1999. `http://www.ics.uci.edu/~eppstein/280/point.html`
11. M. Evans, N. Hastings, and B. Peacock. *Statistical Distributions*. Wiley, New York, 3rd edition, 2000.
12. M. Goncalves, M-E. Vidal. Top-k Skyline: A Unified Approach. In *OTM Workshops*, 2005.
13. M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD*, pages 58–66, 2001.
14. J. Hershberger, N. Shrivastava, S. Suri, and C. Toth. Adaptive spatial partitioning for multidimensional data streams. In *ISAAC*, 2004.
15. T. Johnson, I. Kwok, and R. Ng. Fast computation of 2-dimensional depth contours. In *KDD*, pages 224–228, 1998.
16. V. I. Koltchinskii. M-estimation, convexity and quantiles. *Annals of Statistics*, 25(2):435–477, 1997.
17. A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow distribution. In *ACM Sigmetrics*, 2004.
18. X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting Stars: the k Most Representative Skyline Operator. In *IEEE ICDE*, 2007.
19. M. Muralikrishna and D. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *ACM SIGMOD*, 1988.
20. S. Muthukrishnan. Data streams: Algorithms and applications. In *ACM-SIAM SODA*, 2003.
21. D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1):41–82, 2005.
22. F. P. Preparata and M.I. Shamos. *Computational Geometry : An Introduction*. Springer-Verlag, 2nd edition, 1985.
23. P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. Wiley, 2003.
24. R. Serfling. Quantile functions for multivariate analysis: approaches and applications. *Statistica Neerlandica*, 56(2):214–232, 2002.
25. N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *ACM SenSys*, 2004.
26. S. Suri, C. D. Tóth, and Y. Zhou. Range counting over multidimensional data streams. In *SoCG*, 2004.
27. N. Thaper, P. Indyk, S. Guha, and N. Koudas. Dynamic multidimensional histograms. In *ACM SIGMOD*, pages 359–366, 2002.