Real-Time Exploration of Large Spatiotemporal Datasets based on Order Statistics

Cícero L. Pahins, Nivan Ferreira, and João L. Comba, *Member, IEEE*

Abstract—In recent years sophisticated data structures based on datacubes have been proposed to perform interactive visual exploration of large datasets. While powerful, these approaches overlook the important fact that aggregations used to produce datacubes do not represent the actual distribution of the data being analyzed. As a result, these methods might produce biased results as well as hide important features in the data. In this paper, we introduce the Quantile Datacube Structure (QDS) that bridges this gap by supporting interactive visual exploration based on order statistics. To achieve this, QDS makes use of an efficient non-parametric distribution approximation scheme called p-digest and employs a novel datacube indexing scheme that reduces the memory usage of previous datacube methods. This enables interactive slicing and dicing while accurately approximating the distribution of quantitative variables of interest. We present two case studies that illustrate the ability of QDS to not only build order statistics based visualizations interactively but also to perform event detection on very large datasets. Finally, we present extensive experimental results that validate the effectiveness of QDS regarding memory usage and accuracy in the approximation of order statistics for real-world datasets.

Index Terms—Data structures for visualization, order statistics, quantile sketch, visual analytics, event detection.

1 Introduction

A fundamental problem in modern visual data analysis is how to build data exploration environments that support interactive exploration of large datasets.

This problem has two opposing facets. From one side, the ever-growing complexity and size of datasets bring the need to provide complex navigation and visual summarization capabilities. On the other hand, human perception and cognition pose a challenge on how long the data handling and rendering loop can take. Even small delays on the scale of half a second can have a significant negative impact on the visual data exploration process [1]. Unfortunately, the ability to produce compelling visual summaries, interaction mechanisms, and interfaces has surpassed our capabilities to create techniques that support real-time data processing for visualization [2]. As a result, there are limitations on the analysis that one can hope to perform interactively. In this paper, we are concerned with the scenario of performing real-time analysis (*i.e.*, virtually immediate results) of large static datasets.

Recent efforts propose sophisticated implementations of precomputed indices [3], [4], [5] that store aggregations of a given dataset as solutions to this problem. One limitation of these approaches is the fact that they do not take into account the inherent *distribution uncertainty* due to aggregation: datasets with equal mean and covariance, but with entirely different underlying distributions. Examples of this issue can be seen in the classical Anscombe's Quartet datasets and the work of Matejka et al. [6]. The state-of-the-art method Gaussian Cubes (GC) [7] supports interactive data modeling by describing the data distribution using parametric Gaussian distributions. Unfortunately, this approach has two drawbacks. First, it relies on non-robust statistics (mean and covariances), i.e., they can be easily affected by outliers. Second, and most importantly, one can not assume real-world data to be normal, and assuming normality can hide essential features of the data.

Contribuitions. To overcome these drawbacks we propose Quantile Datacube Structure (QDS): a novel data structure that encodes data distributions based on robust statistics while providing support for interactive visual exploration of large spatiotemporal datasets. To achieve this, QDS couples a non-parametric distribution modeling technique called p-digest, based on the t-digest *quantile sketch* [8] (Sec. 4), with a novel indexing structure that reduces the large memory footprint common to datacube structures and enables real-time slicing and dicing. QDS (described in Sec. 5) extends the querying abilities of previous approaches by supporting queries with order statistics related aggregations such as quantiles and cumulative distribution. We used QDS in a prototype visual analytics system to demonstrate that these queries provide a powerful tool to interactively build widely used visualizations (such as box plots, equi-depth histograms, and band plots), create new ones (such as the heatmaps based on quantiles and cumulative distribution) (Sec. 6) and to perform interactive event detection (Sec. 7). Fig. 1 illustrates interesting spatiotemporal patterns in the distribution of flight arrival delays for U.S. airports found using QDS. Finally, we provide extensive experimental results (Sec. 8) that show the effectiveness of our method for the analysis of real-world datasets scenarios.

2 Related Work

In this section, we review related research on different aspects that play an essential part in this work.

Visualization of Data Distributions. The visualization of statistical summaries is at the core of visual data analysis

• *C. Pahins and J. Comba are with Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Brazil. N. Ferreira is with Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE), Brazil. E-mail: {capahins,comba}@inf.ufrgs.br, nivan@cin.ufpe.br*

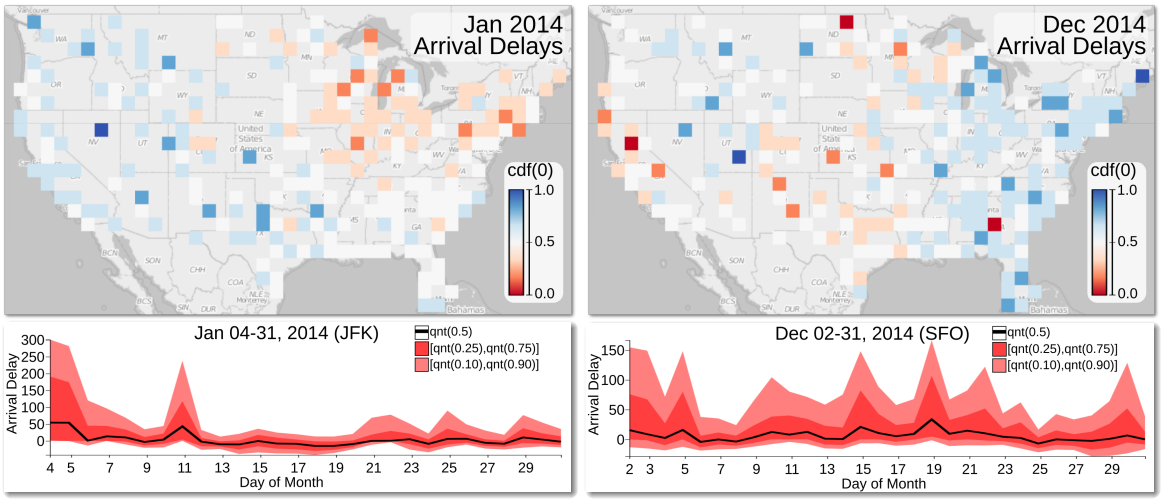


Figure 1. Analyzing the distribution of flight arrival delays for U.S. airports using QDS. We observe two maps showing the probability of flights being late for January and December 2014. Airports are colored using a divergent color scale representing the cumulative distribution function of the arrival delays at the value 0. We assign red color shades to airports with a higher probability of having late arriving flights and blue shades for airports in which flights are more likely to be early. Notice how the trend changes from more likely delayed flights on the Northeastern airports in January to Southwestern ones (particularly in California) in December. The pattern of delay in January 2014 is due to the snowstorms that pounded the Northeast of the U.S. in January. The Western delay pattern in December is due to the so-called "California's storm of the decade" that affected the region in the middle of December 2014. The temporal band plots on the bottom show the evolution of the arrival delay quantiles (0.1,0.25,0.5,0.75,0.9) for both the JFK (left) and SFO airport (right). Dates with a substantial increase in the median arrival delays (black line) are the peaks of these events (e.g., January 4 on the left and December 15 and 19 on the right).

and visual data communication [9], [10]. The most common approach relies on visualizations of the mean and standard deviation such as bar charts and error bar plots. This approach is dubious, sensitive to outliers and may not only introduce bias but also hide essential features of the data (as illustrated in Fig. 2). For these reasons, this approach has been discouraged by researches in the fields of visualization [11], neuroscience [12] and biology [13]. Scientific publications also incentive the use of more accurate distribution representation such as boxplots [14] to summarize large datasets [15]. Furthermore, recent studies by Kay et al. [16] and Fernandes et al. [17] showed that presenting detailed distribution information improves decision making compared to scenarios where this information is not present. In addition, these studies showed that specialized visual summaries based on order statistics improved decision making in an uncertainty judgment in a transit scenario. Our work builds on these observations and proposes a data structure that provides accurate distribution approximations for large spatiotemporal datasets.

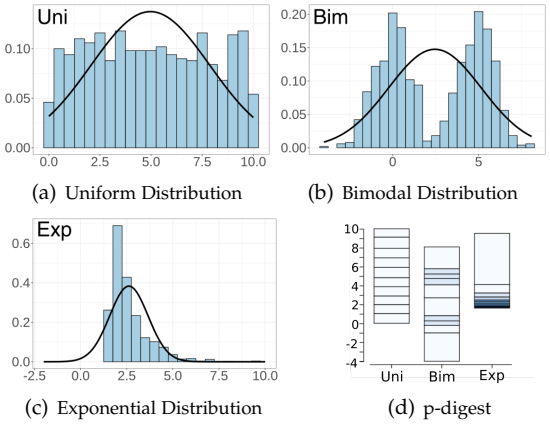
Interactive Visualization of Large Datasets. The problem of providing interactive analytics and visualization for large datasets has attracted the attention of researchers both in the visualization and databases community. Solutions to this problem follow two main strategies: sampling and pre-computation. The sampling strategy uses progressively increasing samples of a population to approximate/estimate the result of a given query [18]. The survey by Chaudhuri et al. [19] describes several techniques for query estimation and data handling in this scenario. In systems using the sampling strategy, users face evolving visualizations that indicate current estimates and, possibly, the uncertainty inherent to the estimation process [20]. While flexible compared to the precomputation strategy, the understanding of the user experience in this scenario is still incipient [21], thus motivating new visualizations and interactions to support users in analytical environments [22], [23].

On the other hand, the pre-computation strategy relies on computing aggregations over several dimensions following the datacube concept. Systems such as Immens [3], Nanocubes (NC) [4] and Hashedcubes (HC) [5] were proposed to reduce the huge memory footprint, but are limited to provide results in counting queries. Recent systems such as TopKube [24] and Gaussian Cubes (GC) [7] extend ordinary datacubes to perform more complex analysis in real-time while respecting reasonable memory constraints. QDS also follows the datacube approach. However, we relax the requirements of exact representation from previous systems to provide a non-parametric approximation of the data distribution. A recent work by Peng et al. [25] proposed a hybrid approach that mixes the sampling and precomputation strategies. However, neither this work or the ones cited above support the quantile queries provided by QDS.

Applications of Event Detection in Visual Analytics. Statistical techniques can be used to identify events or anomaly situations, which has been shown to be a powerful tool for visual anlytics [26]. Maciejewski et al. [27] couple visual exploration with modeling strategies to find abnormal spatiotemporal hotspots. Wilkinson et al. [28] use a statistical algorithm for detecting multidimensional outliers. QDS provides a powerful and flexible way to find relevant and complex events using quantiles from the distributions of large datasets.

3 Background

We briefly discuss the background of probability theory and data sketches, and refer to Rosenthal [29] and Cormode et al. [30] for a detailed description. We define the cumulative distribution function (*cdf*) of a random variable X by $F_X(t) = Pr(X \leq t)$. Quantiles are landmark values of a given *cdf* that define specific points where F_X has



(a) Uniform Distribution (b) Bimodal Distribution (c) Exponential Distribution (d) p-digest

Figure 2. Gaussian distributions are the most common approach of modeling data for analysis and visualization. While this method has theoretical advantages, real-world data is rarely normally distributed. As we observe in (a)-(c) modeling data with normal distributions (black curves) can introduce biases and hide essential features such as multimodality and skewness. As illustrated by the equi-depth histograms produced using p-digest in (d) (darker shades of blue represent higher data density) can efficiently describe the distributions of the other plots.

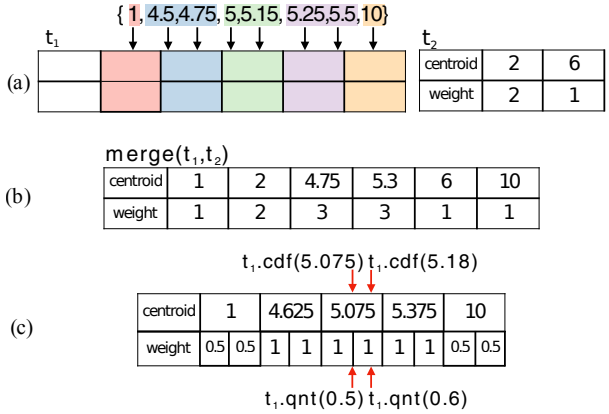


Figure 3. The t-digest sketch: (a) construction of a t-digest t_1. The *cdf* of the input dataset is represented by a set of weighted centroids. (b) Different quantile sketches $t1$ and $t2$ can be combined using the merge operation. (c) A quantile query, qnt(value), interpolates the centroid weights compared to the fraction of the total weight defined by the input value to compute the estimate of the result quantile.

accumulated a fraction of its total probability. For example, a value t is the q^{th} quantile of F_x if $F_X(t) = q$. Intuitively, one can obtain the value of the q^{th} quantile by $F_X^{-1}(q)$ by simply inverting the *cdf*. In this presentation, we focus on the intuition and overlook the fact that *cdf*'s are not necessarily invertible. We define the first (q_1), second (q_2) and third (q_3) *quartiles* as the quantiles that divide the density in four equal parts, *i.e.*, 0.25^{th}, 0.5^{th} and 0.75^{th} respectively. We define a *random field* as a function F_M that associates to each point in a spatial domain (*e.g.* geographical coordinates) a random variable. We define quantile heatmaps and outlierness queries supported by QDS (Sec. 5.1) using random fields.

Unlike moment statistics, such as average and variance, quantiles are robust to the presence of outliers [28]. However, it is not possible to combine quantiles of different datasets (e.g. *cdfs*) without processing the input datasets entirely. This limits the use of quantiles in scenarios that require hierarchical/dynamic aggregation such as datacubes. An alternative is to use approximation schemes called *quantile sketches* [31]. A data sketch is "a data structure that can be easily updated with new or modified data and supports a set of queries whose results approximate queries on the full dataset" [31]. Quantile sketches are data sketches that support queries of quantile and *cdf* estimation. Methods vary in memory usage and approximation performance, leading to two groups of methods. The first one has sketches that have proven approximation bounds such as the proposals of Shrivastava et al. [32], Agarwal et al. [33], Karnin et al. [34] and Felber and Ostrovsky [35]. Such methods have performance requirements which incur in complex algorithms that use large amounts of memory in practice (see discussion in [36]). The second group of methods lack rigorous algorithmic analysis but relies on heuristics to provide empirical results for query accuracy and reduced memory usage. Examples of methods in this group are the GK sketch [37], the S-Hist sketch [36] and the t-digest by Dunning [8].

4 The t-digest data sketch

The simplicity and approximation accuracy of t-digest singles it out from other quantile sketches. The t-digest summarizes the (empirical) *cdf* of an input dataset by a set of weighted values called centroids (Fig. 3). To choose centroids we group elements on subsequences of varying size following an adaptive strategy. Given an input *compression* parameter δ that defines the maximum number of centroids, the strategy gives high priority to extreme quantiles (closer to 0 and 1), as defined by the function $k_\delta(q) = \delta((sin^{-1}(2q-1)+\pi)/2\pi)$. The size of each subsequence is smaller (i.e., more resolution) for centroids near the beginning or the end of the dataset, but larger towards the middle. This strategy tries to make queries for extreme quantiles, in general, more accurate than the ones closer to the median for outlier detection purposes. The construction process of t-digest, illustrated in Fig. 3(a), is closely related to the process of merging two sketches (Fig. 3(b)). The construction of one sketch requires merging a dataset (the elements correspond to centroids with weights equal to 1) against an empty t-digest. This process consists of sorting the weighted centroids and performing the grouping of subsequences as before but considering the given weights. To perform the query for a quantile we divide the weight of each centroid into two equal parts to the left and the right of the centroid. The quantile query receives the desired q and loops through the ordered list of centroids accumulating all the weights that have already been seen and comparing it to $q * |D|$, where $|D|$ represents the sum of weights (size of the dataset). If the desired weight ends up on a centroid, the value of that centroid is returned. This happens in the median query $qnt(0.5)$ in Fig. 3(c). On the other hand, if the weight ends up between two centroids, the value of the quantile is derived by linearly interpolating the values of the corresponding centroids using their weights (e.g., $qnt(0.6)$). The *cdf* query is implemented as the inverse of the result of a quantile query. Counting queries are also supported and return the sum of the weights of each centroid.

The publicly available implementations of t-digest were designed for applications in a data streaming scenario with low memory constraints. Their large memory overhead makes them not adequate to be used in datacube structures. We propose an optimized method called p-digest that reduces the memory footprint of the previous implementations and, therefore, suitable for our applications.

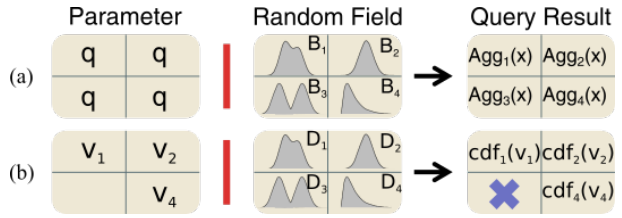


Figure 4. Queries supported by QDS. Let F_M be the random field formed by merging quantile sketches of selected bins. (a) The quantile and cdf queries receive a parameter x and returns the result of the corresponding query for each quantile sketch. (b) The pipeline query: we use the result of a given query as a parameter to a second one using a right join process. In case the parameter has "missing" bins the result query can have undefined (purple X) values.

5 Quantile Datacube Structure

In this section, we describe the Quantile Datacube Structure (QDS). We present the queries supported, its internal representation, query algorithm, and implementation details.

5.1 Overview and Query Types

Consider, for example, a hypothetical scenario of the analysis of flight delays in U.S. airports. We are interested in answering questions like T_1:*"How likely is a flight operated by Delta Airlines to be delayed more than 10 minutes at JFK airport?"*, T_2:*"How does the distribution of flight delays for two airports compare to each other in the past month?"* and T_3:*"How unusual are the delays experienced by Delta flights on January 29th, 2017?"*. To answer such queries QDS stores a quantile sketch as a payload at each node of a datacube to allow fast data selection and accurate querying for distribution statistics. QDS supports the following primary type of query:

select AGGR from QDS where CONSTRAINTS [group BY G]

The CONSTRAINTS part of the query represent conditions defined on any set of the *index dimensions* (e.g., carrier=Delta, airport=JFK) and specify the datacube nodes to consider. QDS groups these nodes into bins according to the group by dimension G (or create one group for all nodes if this optional information is not given). The quantile sketches associated with each datacube node in each group are merged to represent the distribution of the data in each group. In case a group by dimension G is specified, we merge sketches according to the bins of D forming a random field. For example, in our flight's scenario, grouping by the airport dimension will result in a collection of p-digests associated with each airport. Similarly, grouping by a temporal dimension with a given time granularity will result in a p-digest associated with each timestamp. The same is valid to spatial grouping (e.g., map tiles with resolution = 8 produces a maximum 256x256 p-digest per tile). The aggregation function (AGGR) is executed on a *measure dimension* (e.g., arrival delay) and defines the quantile sketch query we execute on the random field: *quantile, cdf or count*. This process is illustrated in Fig. 4 (a).

Quantile and *cdf* queries can answer questions T_1 and T_2 above. To answer question T_3, we use another query called *pipeline* (Fig. 4 (b)), which use the output of a query as input to a second one. In the T_3 example, we perform the first query to select all flight delays for Delta on January 29th. Let a second query select the total distribution of flight delays by Delta. QDS performs an operation of *right join* between

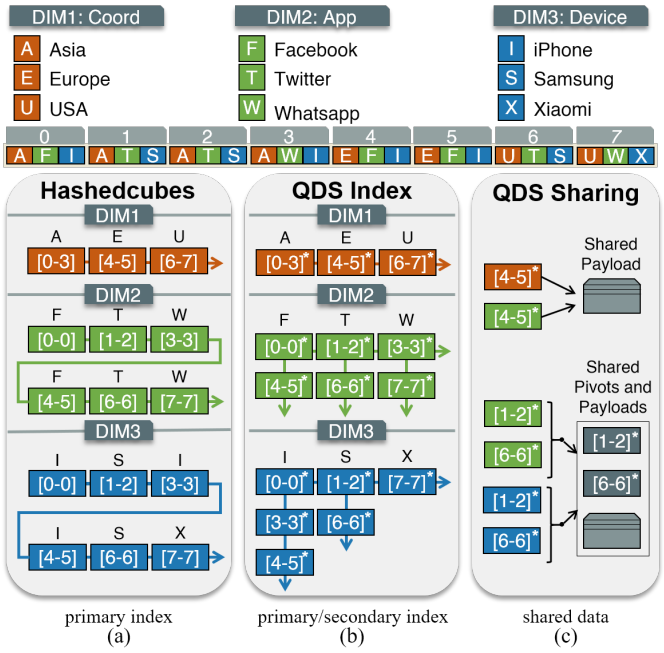


Figure 5. QDS indexing scheme and shared pointers. We use an example to compare the indexes of HC (a) and QDS (b). The input dataset has eight records, each with three dimensions. In HC, each dimension stores pivots in a pivot array that refers to intervals in the input dataset. In QDS, in addition to the pivot array for each dimension (primary pivot array), we keep a secondary pivot array for each element. In graphical terms, the primary array is displayed horizontally, while the secondary array is displayed vertically. Searching for values equal to F in QDS can be simply done by following vertically the secondary pivot array associated with F in dimension 2. The number of pivots stored in the QDS is not larger than in HC. Each pivot has an additional payload (marked with *) that can store quantile information. (c) Pivot arrays tend to have duplicate information across dimensions. To save memory, QDS used shared pointers to compact shared pivot and payload information.

the bins resulting from these two queries and compute the aggregation of the second query for each value in the output of the first query. The result is a score quantifying the *cdf* for January 29th in each airport. As described in Sec. 6.3, pipeline queries are the base for our event detection method.

The last query type supported by QDS is used to quantify the total deviation from the median over a period of time. Given a start/end timestamp and a temporal resolution (e.g., days) this query performs a set of pipeline queries for each timestep. In each timestep, the values are added up to create a score for each temporal bin. We name these *composite queries* and give examples in the use cases of Sec. 7. A detailed description of the execution of the query algorithm is presented in Sec. 5.3 and illustrated in Fig. 6.

5.2 Internal Representation

Datacube-inspired structures have as a common challenge the need to store data as compressed as possible while supporting fast query response. The exploration of data with order statistics creates additional challenges. We describe below the indexing scheme, compression of shared information, and p-digest sketch that stores quantile data.

5.2.1 Indexing Scheme

The design of QDS is inspired by Hashedcubes (HC) because it offers the best trade-off regarding storage and

efficiency. Since both structures have similar concepts, it is important first to review the design of Hashedcubes. To do so, we will use a simple dataset containing eight records (labeled from 0 to 7), each containing three categorical dimensions (location, app, and device) shown on the top of Fig. 5. Following a pre-defined ordering of the dimensions of the input dataset, HC keeps a multi-level index. For each dimension, this index stores an array of pivots that delimits a consecutive interval in the sorted input array with equal values. Fig. 5(a) shows an example of the index (pivot arrays) created with our sample dataset. In dimension 1 (location), one entry in the array has a pivot $[4-5]$ associated to the value E (Europe), meaning that in the input array, entries from 4 to 5 have values E in the first dimension. Observe that at dimension two there is more than one pivot associated with the values F (Facebook), T (Twitter) and W (Whatsapp). As a result, a query for F in the second dimension must find all its non-contiguous pivots. This is a simple example of *pivot fragmentation* which is a consequence of the multi-key sorting of pivot arrays in each dimension. As more dimensions are used, fragmentation increases, which causes queries that use subsequent dimensions to examine a possibly considerable number of pivots We experienced this corner case when implementing HC (see Sec. 8).

QDS's novel pivot index (Fig. 5(b)) fixes the fragmentation issue as well as supports the varied set of queries described previously. Starting at the second dimension, instead of a single pivot array, we keep an additional *secondary pivot array* that can be used to recover all pivots associated with a given value. For example, searching for values equal to F in the second dimension can be done by following the secondary pivot array associated with values F, which return the pivots [0-0] and [4-5]. The secondary pivot array allows keys associated with pivots to be stored only once, thus saving memory. Another improvement is related to the fact that pivot arrays for distinct dimensions in HC often have duplicated entries, leading to redundant storage. QDS overcomes this problem with a shared container abstraction. For example, in Fig. 5(b) we have the pivot [4-5] appearing in dimensions 1 and 2. Using a shared abstraction we create a single payload that is shared for both pivots, as show in Fig. 5(c). A second, and more sophisticated sharing happens when the secondary array is identical for different dimensions. In Fig. 5(b), the secondary array associated with the value T in the second dimension has pivots [1-2] and [6-6]. Similarly, the secondary array associated with the value S in the third dimension also has pivots [1-2] and [6-6]. In such cases, we share both the payload as well as the pivots, as shown in Fig. 5(c). We refer to Sec. 8 for experimental results of memory saved by these optimizations.

5.2.2 The p-digest data sketch

The t-digest described in Sec. 4 was our choice for storing payload information because it supports compressed and accurate on-line order statistics. There are, however, limitations in the two publicly available implementations of t-digest. The main implementation, described in [8], uses a balanced binary search tree (AVL) to store centroid information, consuming 80 bytes per centroid. A secondary (and under construction) implementation uses an array, which reduces memory usage to 40 bytes per centroid. Such memory requirements are adequate for the streaming processing applications of t-digest, but in our datacube scenario, it results in prohibitive memory usage.

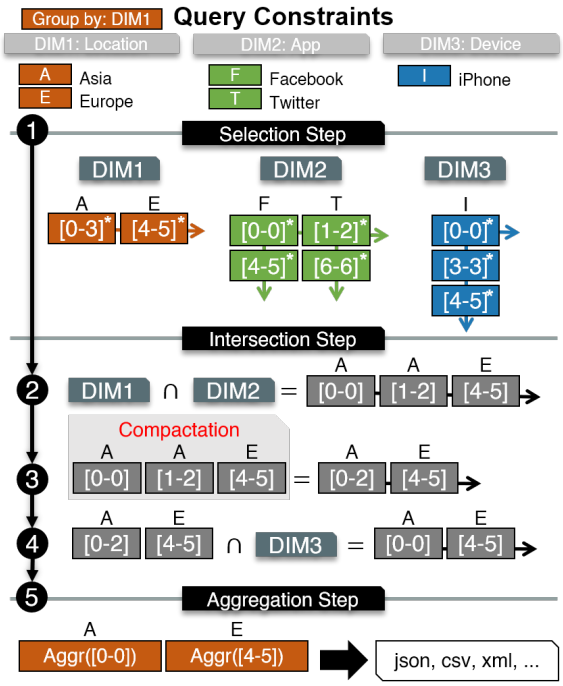


Figure 6. The QDS query algorithm demonstrated using the dataset of Fig. 5. The input query has constraints in all dimensions. In the *selection* step, the pivot array of each dimension is processed to check the pivots that satisfy the query for that dimension. In the *intersection* step, we compute in sequence the intersection of the results of previous steps. The *aggregation* step compacts the results of the previous step.

We made several changes to the array implementation of t-digest to comply with our performance requirements, For instance, we reduced the centroid memory storage by implementing the sketch as a stream of numbers, with both $centroid$ and $weight$ arrays as a single chunk of floats. The memory requirements for the centroid is at most 8 bytes, using 4 bytes for each of the centroid and weighted arrays. Using QDS with real data, a situation that frequently occurs is the weight array have all values equal to 1. To leverage this property and reduce memory usage, we added a boolean field to the end of the payload structure to indicate the storage of both centroid and weighted arrays. When this field is 0, the weight values are all equal to 1, and weights are not stored explicitly, only the centroid values. On average the cost for centroids is just 4 bytes. Similarly, we do not store the $weight$ array when all its values are equal. Such optimization is efficient for (very) small pivots in deeper dimensions. The $merge$ and $query$ operations were modified to work with this modified structure. For convenience, we call this modified structure by the name *p-digest*, since in QDS it associates one such sketch to each pivot. We implemented p-digest as a standalone library that can also be used outside QDS, which is available as an alternate implementation of t-digest (Sec. 5.4).

5.3 Query Algorithm

While QDS's and Hashedcubes's indices use similar concepts, their structural differences and the sophisticated set of queries supported by QDS makes querying our structure

a very different process. QDS's query algorithm (Fig. 6) is responsible for efficiently selecting nodes and satisfying a set of query constraints. This algorithm was designed to handle a great variety of query combinations following a progressive refinement approach. In a high-level description, for a given multi-dimensional query, the query algorithm is composed of three steps executed in sequence: *selection, intersection, and aggregation*. In the *selection step*, for each dimension specified in the query, the algorithm selects the pivots from the pivot arrays that satisfies the query individually for each dimension. The primary and secondary pivot arrays are responsible for efficiently discarding queries that return empty results, thus avoiding the HC corner cases mentioned before (see a discussion on Sec. 8). The *intersection step* is responsible for combining the pivots, resulting from the selection step, that simultaneously satisfies the query for all dimensions. The result of the intersection step are pivots that might not be contiguous since the previous steps might leave similar elements distributed over distinct pivots. The *aggregation step* groups pivots by compacting disjoint pivots that contiguously represent the same value. For example, if pivots [1-2] and [3-5] refer to the same value F, we replace by a single pivot [1-5] of value F.

5.4 Implementation

QDS is implemented in C++ and uses a client-server architecture. The server consumes an input, and builds a QDS in a pre-processing step. QDS supports multiple categorical, temporal and spatial dimensions for its indexing schemas. We discretize spatial and temporal dimensions like in NC or HC: quadtree based map tile coordinates and user-defined time bins, respectively. Unlike NC and HC, QDS can stack and intercalate different types of dimensions without a predefined order (e.g, categorical-temporal-spatial, or even, NC and HC ordering of spatial-categorical-temporal), since it impacts both memory usage and running time. Note that QDS default layout is the inverse of both NC and HC, since we find this to be a good compromise between performance and memory usage (refer to Sec. 8). We also support multiple measure dimensions by storing unique combinations (i.e., pivots) into individual primitive arrays (referenced as *payloads*). Each payload uses a header to determine the beginning and end of each dimension. Other low-level QDS optimizations are accessible in its open source code available at https://github.com/cicerolp/qds.

6 Building Visualizations with QDS

We illustrate below general scenarios of analytical tasks and visualizations enabled by QDS to support the visual analysis of large datasets with order statistics data.

6.1 Extending Usual Visualizations

The query capabilities of QDS support the analysis of data distribution patterns in spatial, temporal and categorical dimensions. For example, we define *quantile heatmaps* as heatmaps obtained from QDS's quantile queries. Fig. 7 compares the standard mean heatmap (a) against quartile queries (b,c,d) of taxi trip fares (in US dollars) in NYC

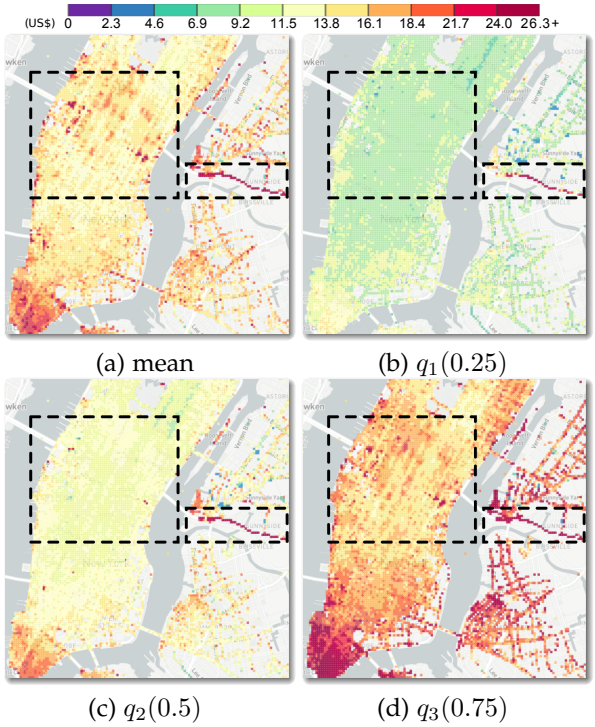


Figure 7. Quantile heatmaps of taxi trip fares in NYC during the month of October 2014 based on their pick-up locations. The mean based heatmap (a) conveys high prices similar to the third quartile map(d). The median heatmap shows lower fares (c) indicating the robustness with respect to outliers. The first quartile map (b) indicates mostly lower values except on regions close to the Queens–Midtown Expressway near the high traffic region of Queens—Midtown Tunnel's toll station (right dashed box).

based on their pick-up locations. We notice how the quantile heatmaps convey a different message than the mean heatmap. While the mean map (a) suggests high prices (above 16 dollars) for the region of Midtown (left dashed boxes), both maps of the first quartile (b) and median (c) suggest that these prices are usually smaller in that region (below 14 dollars). Also, notice how the mean map is similar to the third quartile map (d). This reflects the sensitivity of the mean to outliers. Furthermore, by performing a simple arithmetic operation on quantile heatmaps, we visualize spatial properties of the underlying distributions such as interquartile range (a robust alternative to variance as a measure of spread/uncertainty) and skewness (a measure of asymmetry in the distribution) [38]. Another novel notion of heatmap enabled by QDS is called *cdf heatmaps*. These maps use the *cdf* query to display how likely a distribution in a given location is to be smaller than a certain value. Fig. 1 shows examples of this concept. The color on the maps represent the probability of flights being late, *i.e.*, $cdf(0)$. These maps make it intuitive to observe the changes in the geographical delay pattern from east to west in 2014.

The temporal aspect of quantiles can be explored for example by constructing *band plots* (Fig. 1 bottom). The median (black) curve gives a robust notion of centrality and therefore the typical temporal behavior of the variable in consideration. The curves of the first quartile (bottom curve) and third quartile (top curve) form dark red bands. The lighter red band is formed by quantiles 0.1 (bottom) and 0.9 (top). This choice was made to avoid minimum and maximum outlier values. Notice how the additional

quantiles help in the identification of the variability of the distribution. Finally, we also notice different forms in the average and error bar based temporal plots (produced by GC [7]). The bands formed are not necessarily symmetric around the median curve and are a more faithful representation of the distribution behavior over time.

As the last example, QDS can be used to understand the distribution of quantitative values related to categorical dimensions. In fact, this can be done by using the quantile information to build the widely used box plots (Fig. 8) and *equi-depth histograms* (Fig. 2(d)). Unlike conventional histograms, the bins in equi-depth histograms contain a fixed fraction of the data population (equally spaced quantiles). In Fig. 2(d) the bins are colored proportional to their data density to better depict the data distribution.

6.2 Easing the Reading of Uncertainty Visualizations

Interpreting uncertainty visualizations is not an easy task, even for trained individuals, One issued for this difficulty is performing statistical inferences by eye to quantify the uncertainty related to analytical tasks. An example of such scenario can be seen in Fig. 8. How likely is it for each of the distributions to be smaller than the red line (which represents the threshold of considering a flight to be late)? To simplify this problem, Ferreira at al. [22] proposed interactive annotations that enrich usual uncertainty visualizations by visually quantifying uncertainty. One of these annotation allows the user to drag a line and the likelihood of the distribution to be smaller than this line would be mapped to colors. They provide a user evaluation that indicates the effectiveness of annotations in the sense that it improves "justified confidence", *i.e.*, the correlation between the user being correct and being confident her answer. However, Ferreira at al. [22] did not propose an efficient data handling method to support these interactions. In fact, they used an ad-hoc sampling scheme which neither scales with the number of distributions nor supports slicing and dicing. Therefore it can not be used in a real visual analytics system. QDS can be used to support the interaction described above: it suffices to use the *cdf* query in each box plot with the value represented by the red line as parameter. The results of these operations are used to color the box plots in Fig. 8.

6.3 Uncovering the Unexpected

Performing visual exploration on large amounts of spatiotemporal data can be a time-consuming process. In fact, due to the inherent complexity of this data unusual (and possibly interesting) patterns might occur at multiple aggregation scales and therefore finding them requires users to inspect a large number of data slices over time and space. Thus, these patterns might remain undiscovered even after the use of visual exploration tools [26]. For this reason, the application of event detection techniques is essential to find these patterns. QDS's ability to retrieve the (approximate) distribution to an arbitrary portion of the data interactively is a powerful tool to perform event detection in a visual analytics system. In fact, given a value t and the distribution function F_X of a random variable X, we can define a measure of outlierness of t concerning F_X as [39]: $\hat{\phi}(t, F_X) = 2(|0.5 - F_X(t)|)$. A low value of $\hat{\phi}(t, F_X)$ means that t is a "normal" data instance, while high values mean instances closer to extreme values of the distribution and therefore judged as "events". Fraiman and Muniz [39] proposed a method to extend this measure of outlierness to higher dimensional data. To describe this extension, we use as an example a heatmap m (analogous to t in the unidimensional case) of prices of taxi trips similar to the example given in Fig. 7(a). The function m assigns, for each geographical location p, a value $m(p)$, corresponding to the average price of taxi trips starting from that location. We assume to be given the random field F_M of the prices of taxi trips for every geographical location (analogous to F_X). We define the outlierness of m concerning F_M as $\phi(m, F_M) = \int \hat{\phi}(m(p), F_M(p))dp$, where the integral is taken over all points p on the map domain, and $m(p)$ and $F_M(p)$ denote the value of the heatmap m and the distribution of fare values at location p respectively. We compute the value of $\phi(m, F_M)$ using QDS's pipeline query described in Sec. 5.1. We choose the geographical coordinates as the group by dimension. In this manner, we can perform a *cdf* query for each $tile$ on the map. To obtain the final result, we simply compute $\hat{\phi}$ on each of these values and add up all the results. Such an approach can be used to find events in datasets with a long temporal range (Fig. 9).

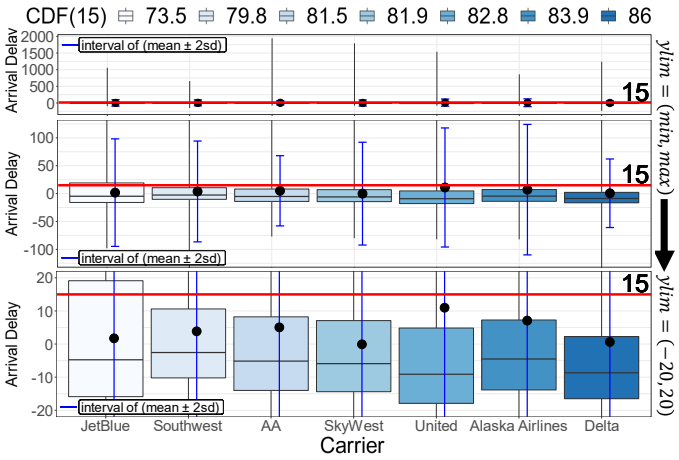


Figure 8. Box plot of flight arrival delays per carrier. The boxes are colored and sorted according to the probability of each distribution being below 15 minutes (red line), which represents the proportion of on-time flights.

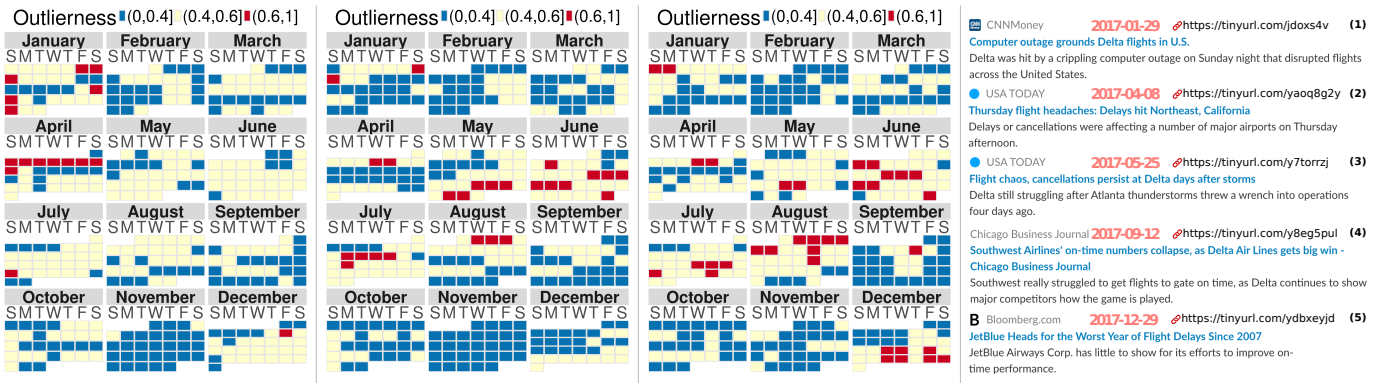
7 Use Cases

We demonstrate the capabilities of QDS in real exploration scenarios. We obtain all analyses while exploring datasets with millions of records interactively in a prototype visualization system using QDS, as can be seen in the demo video[1]. In all use cases we used p-digest's compression parameter $\delta = 50$ for the QDS construction.

7.1 Analyzing Flights Delays

Delayed flights have a large impact on the finances of air carriers. According to the trade group *Airlines of America* each minute of delay costs around $62.55 to U.S. airlines [40]. Such costs represent a significant loss considering that some airlines accumulate millions of delay minutes every year.

1. https://youtu.be/WSzTJXIVUw4



(a) Delta (b) JetBlue (c) SouthWest (d) Flight delays news

Figure 9. Daily arrival delay outlierness in 2017 for Delta JetBlue and Southwest airlines. Delta had an abnormal first week of April due to severe weather in its hub city Atlanta. Similarly, weather events created abnormal arrival times for JetBlue and Southwest in May and August respectively. January 29th is another odd day: a computer outage grounded all Delta's flights. The news on the side corroborate the unexpected events found.

The *On-time Performance* dataset made available by the U.S. Department of Transportation [41] tracks the delays of U.S. air carriers domestic flights. This dataset has over 178 million flights in 30 years (1987 to 2017). To analyze this data we built a QDS structure on 9 of the 29 original columns of the dataset. As part of the index scheme, we used the categorical dimensions *canceled, diverted, carrier, departureAirport*, the temporal dimensions *departure time*, and *latitude and longitude* as the spatial dimension. We used the *departure delay* and *arrival delay* as payload dimensions.

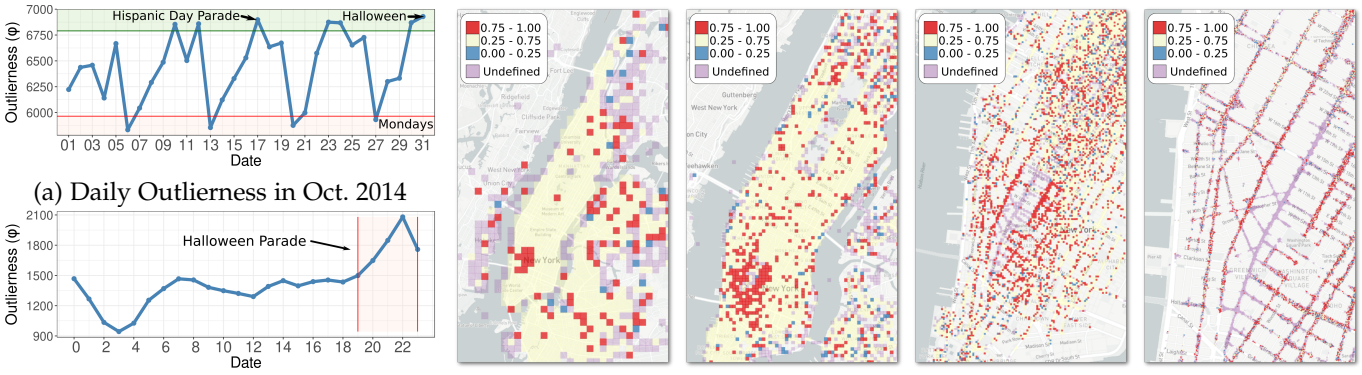
The *U.S. Bureau of Transportation Statistics* (USBTS) publishes periodic reports of carrier on-time performance. In these reports, a flight is on-time if it arrives no later than 15 minutes of its scheduled time. Fig. 8 shows a box plot of flight arrival delays distributions for some U.S. airlines, obtained using the QDS, using the data from the January 2017 through October of the same year. We use the sliding line interaction described in Sec. 6.2 to quantify the proportion of delayed flights according to the 15 minutes threshold. The results of $cdf(15)$ are used to color the boxes in the box plots. We also used them to sort the boxes in ascending order, to rank the airlines according to their delays. An interesting observation is a bad performance represented by the company JetBlue, for which 2017 was the worst year of flight delays in the previous decade (Fig. 9(d)-5). Also, we highlight that the ranking of carriers obtained by QDS matches the one reported by the USBTS for the period and the inferred densities are very close to the ones reported [42].

We study the JetBlue, Southwest, and Delta airlines to understand events that affect their delay patterns. For each company, we use QDS's outlierness query (Sec. 6.3) to quantify how unusual one day is if compared to the distribution of delays over the entire year. The results of this analysis are presented in Fig. 9. Days colored in red, yellow and blue have high, medium, and low measures of outlierness respectively. An interesting case is the entire red week for Delta at the beginning of April. This odd week for the company was caused by severe thunderstorms that happen in Delta's hub city of Atlanta. During this week more than 3000 Delta flights were canceled (Fig. 9(d)-2). At the end of May, JetBlue had delays due to heavy rains in the Northeast of the U.S, leading to cancellations in main airports for JetBlue in New York City and Boston (Fig. 9(d)-3). Southwest experienced unusual delays in August due to Summer thunderstorms, the Hurricane Harvey, and the high seasonal demand (Fig. 9(d)-4). While weather is the main cause of flight delays in the U.S., we found an equipment malfunction event (a computer outage) that grounded all Delta's domestic flights on January 29 (Fig. 9(d)-1).

7.2 Exploring Outlierness in Taxi Trip Records

New York City (NYC) is one of the largest cities in the world. Its taxi system is a big part of the city's life, with more than 13 thousand cabs driving every day. The NYC Taxi and Limousine Commission have collected and distributed monthly yellow taxi trips records since 2009 [43]. We use QDS to analyze some of the fields in this dataset. The QDS index has pickup location (latitude and longitude) as the spatial dimension, pickup date-time as the temporal dimension and passenger count and payment type as the categorical dimension. As payload dimensions, we use the total fare and trip distance.

We describe interesting events that happened during October of 2014. For each day in that month, we computed the outlierness of the total fare of trips compared to the distribution of the entire month (Fig. 10). We observe that Mondays in that month (days 6, 13, 20 and 27) have the lowest outlierness (shaded red region). On the other hand, days colored in shaded green are on top of the outlier list, corresponding to Thursdays (23, 30), Sundays (12) and Fridays (10, 17, 24, 31). The top of the list a Friday (31). To justify this, we notice, for example, that Sunday 12 was the day preceding the Columbus day holiday on which some events changed the traffic on major avenues most of the day. In fact the CBGB Music Festival blocked a portion of Broadway from 10:30 AM to 19:30 PM and the Hispanic Columbus Day Parade closed a long portion of Fifth Avenue from noon to 5 PM [44]. We explore the top candidate according to our outlierness metric, October 31. To investigate what makes this day stand out, we performed another outlierness time analysis now comparing each hour on this day against the distribution of total fare of all hours in the month. The resulting time-series can be seen in Fig. 10(b). We see that the outlierness attains its highest values at the end of the day starting at 7 PM. To understand what makes this hour to stand out, we use a composite cdf query to map how the distribution of these hours (7 PM to midnight) compares to



(a) Daily Outlierness in Oct. 2014

(b) Hourly Outlierness on Oct. 31 2014 (c) Heatmaps showing composite cdf query results for 31 Oct. 2014 - 7 to 10 PM

Figure 10. Exploration of NYC yellow taxi trips in October 2014. (a) Outlierness coefficients with respect to total fare vary widely during the month with peaks on days 10, 12, 17, 23, 24 and 31. This last one being the highest. (b) Analyzing how the outlierness vary over the day 31 we see that the day got more "unusual" with the highest values on the period starting at 7 PM. (c) The heatmap resulting from the pipeline query in this period we observe from left to right that trips are more expensive than normal in the Greenwich Village Region. Zooming in we see that a portion of the streets (purple) that unusually did not have any trips. This corresponds to the area where the annual Greenwich Village Halloween parade happened.

Table 1
Overall summary of the relevant information for building QDS.

dataset	size	index schema(bits)	payload schema	QDS Memory/Time		HC Memory/Time
				leaf-size = 1	leaf-size = 32 or 64	leaf-size = 32 or 64
brightkite	4.5 M	dayOfWeek (3), hourOfDay (5), time (16), lat (25), lon (25)	NA	455 MB/9s	276 MB/7s	366 MB/7s
gowalla	6.4 M	dayOfWeek (3), hourOfDay (5), time (16), lat (25), lon (25)	NA	711 MB/13s	367 MB/11s	743 MB/13s
twitter-small	210.6 M	device (3), time (16), lat (17), lon (17)	NA	3.1 GB/05:55m	2.7 GB/05:54m	4.9 GB/10:53m
twitter	210.6 M	app (2), device (3), language (5), time (16), lat (17), lon (17)	NA	4.6 GB/06:39m	4.2 GB/06:37m	9.4 GB/12:04m
flights	121.2 M	dep. delay (4), carrier (11), dep. time (16), lat (25), lon (25)	arrDelay, depDelay	1.4 GB/02:50m	1.4GB/02:50m	457 MB/03:56m
green-taxis-small	42 M	pickupDateTime (16), lat (22), lon (22)	ttlAmount, distance	1.3 GB/01:24m	1.2 GB/01:16m	788 MB/03:56m
green-taxis	42 M	dayOfWeek (3), hourOfDay (5), pickupTime (16), lat (22), lon (22)	ttlAmount, distance	1.3 GB/01:16m	1.2 GB/01:15m	3.0 GB/01:49m
yellow-taxis-small	706 M	pickupDateTime (16), lat (22), lon (22)	ttlAmount, distance	9.7 GB/27:53m	9.3 GB/28:04m	7.0 GB/18:14m
yellow-taxis	706 M	dayOfWeek (3), hourOfDay (5), pickupTime (day), lat (22), lon (22)	ttlAmount, distance	9.7 GB/31:37m	9.3 GB/31:33m	12.6 GB/20:38m

the rest of the month. Colors in the map reflect the results of the composite cdf query: blue, yellow, red and purple mean low, medium, high and missing quantile values respectively. Looking at the map of the city (Fig. 10(c)) we see a large red region (trips more expensive than normal) on the Greenwich Village. Such trips have fares 75% more expensive than fares of the entire month. A zoom in this area shows progressively more details of this pattern, revealing that expensive trips happened around an area where no trips happened (purple region) during the interval from 7 PM to midnight. This area corresponds to a portion of the 6^{th} avenue, where the annual Greenwich Village Halloween parade happened in 2014.

8 Experimental Results

We evaluate our method in two sets of experiments. The first one evaluates the QDS index. We begin by performing a direct comparison to HC regarding construction time and memory usage for several datasets and schemas. Later we compare the response time of count queries in QDS as well as to the three commonly used databases SQLite, PostgreSQL and MonetDB. The second set of experiments evaluates the p-digest payload performance concerning approximation accuracy, memory usage, and computational performance. We compare against the quantile query capabilities present in the database solutions previously mentioned. The experiments were performed in a Linux-based machine, with an Intel Core i7 4790 with 32GB of main memory. We used the default options of the databases and the SQLite in-memory configuration. Benchmark data and code are available at the QDS's code repository.

8.1 The QDS Index Experiments

Memory usage. We compare the memory usage and construction time of QDS and HC for different datasets and schemas (Table 1). HC adopts a *minimum leaf-size* in its spatial dimension to improve query time and memory footprint. On the other hand, it leads to poor visual accuracy for regions with a low number of elements and, more importantly, for outliers. To enable a direct comparison with HC, for this benchmark, we build an experimental version of QDS with a modified implementation of both construction and query algorithms that integrate the minimum leaf-size technique. We notice that QDS's memory usage is comparable (and in some cases much better than) with Hashedcubes (*leaf-size = 32 or 64*). Regarding construction time, our method achieved better results than HC in most of the datasets considered even if we do not modify the leaf-size (*leaf-size = 1*).

Query Latency. We assess the performance of QDS's index by measuring its latency on the same 87449 spatio-temporal count queries (with spatial, categorical and temporal constraints) used on the HC paper. These queries were collected on the public NC site while users explored the *brightkite*, *gowalla*, *flights* and *twitter* datasets. We compare the results from QDS, HC and spatial extensions of SQLite [45], PostgreSQL [46] and MonetDB [47], using their recommend approach to accelerate spatial queries. To enable the direct comparison of data cube structures and database solutions, we translated the set of queries mentioned above to the appropriate format of each system. As observed in Fig. 11, QDS outperforms all the tested solutions with query latency typically lying below 10ms. We highlight that it successfully avoids HC corner cases.We notice that SQLite, PostgreSQL and MonetDB spatial indexes implementations were unable

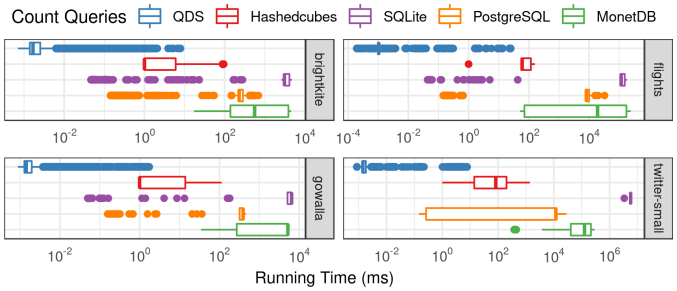


Figure 11. Performance comparison of QDS, HC and database alternatives computing count queries. QDS novel index successfully avoid HC corner cases and offers a query latency that typically lies below 10ms in various datasets.

to offer efficient mechanisms to perform spatial filtering while combining categorical and temporal constraints. We specially notice that MonetDB does not support any special accelerators for spatial objects[2] and hence its poor performance. Overall, QDS's index offers real-time ($< 40ms$) slicing and dicing with ease, which we use to provide complex quantile queries at interactive rates (e.g., pipeline queries).

8.2 The p-digest Sketch Experiments

We now report accuracy, performance and memory usage of p-digest through five experiments. The first three of them evaluate the QDS's memory/accuracy trade-off introduced by p-digest's compression parameter δ. To evaluate this trade-off, we measure the quality of quantile estimation in different conditions of data compression, merge effectiveness and queried quantile, while varying δ. For this experiments, we used the *green-taxis* dataset to stress p-digest worst-case scenarios. The values shown in the accuracy experiments are computed by measuring the relative error of estimated quantile to the actual empirical quantile for the input data: $|q_{estimated} - q_{empirical}|/(|q_{empirical}| + 1)$.

Accuracy per Spatial Quadtree Node. This experiment gives an insight into the accuracy of p-digest when dealing with input data that range from 1 element to 100 million elements. After loading the dataset into QDS, we query each region of the spatial index independently from `root` to `leafs` at height 25, measuring the accuracy during the process. This benchmark exploits p-digest capacity to approximate different set sizes. Fig. 12 (a) shows that quantile estimation is accurate for small input data and nearly unaffected for variations on compression parameter δ. The average error is somewhat constant for larger inputs.

Accuracy per Spatial Quadtree Level. In this experiment, we combine each quadtree level into its respective p-digest, i.e., for every level, we execute (at most) 4^Z p-digest merge operations while keeping the same input data. Fig. 12 (b) shows that accuracy increases when the input data is broken into more parts, because data is spread across more p-digest arrays. QDS benefits from this since pivot intersections (and as a result, p-digest merges) are commonly executed to answer the range of queries we support. Compression parameter $\delta = 50$ was the default value because it has the right balance between (i) accuracy per number of elements, (ii) accuracy per number of merges and (iii) memory usage.

2. https://www.monetdb.org/Documentation/Extensions/GIS.

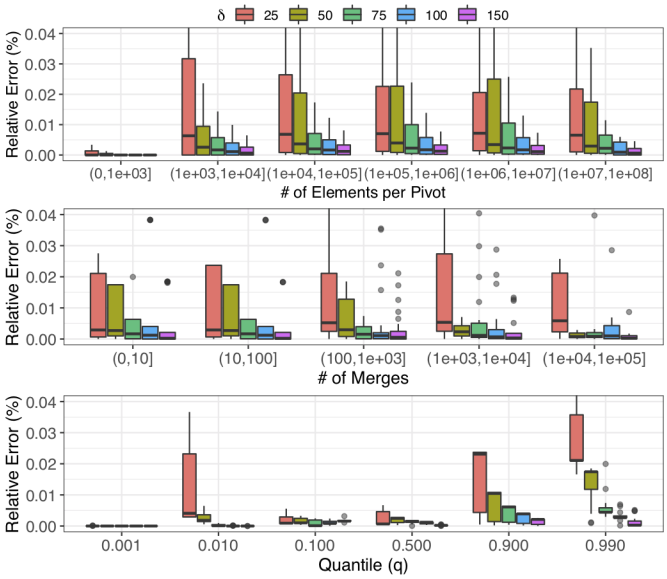


Figure 12. Evaluation of p-digest's quantile estimation with respect to (a) pivot size, (b) number of merge operations and (c) queried quantile.

Accuracy by Varying $q^{th} \in (0, 1)$. To measure accuracy on extreme quantiles, we merge each quadtree level into its respective p-digest and aggregate the estimated quantiles per q^{th} (Fig. 12 (c)). This experiment gives an insight into the error of a typical real-world query and the importance of the choice of the compression parameter δ. The relative error of compression parameter $\delta = 25$ gets worse near $q = 1$. This behavior reflects a poor choice of this parameter for the distributions that we are trying to approximate. Notice how the performance improves for larger values of δ.

Performance. We now compare the latency of QDS quantile queries against similar queries provided by SQLite, PostgreSQL and MonetDB. As a baseline for evaluation of p-digest, we also implemented an experimental variaion QDS, here referenced as QDS (w/o p-digest), that performs exact quantile computation. To do so we use QDS index and store at each pivot a sorted array containing the corresponding payload t values. This baseline give us an insight about the performance of QDS index when using a naive approach to calculate quantiles.

For this experiment we use a synthetic dataset composed of 50 million points (x, y, t), where the spatial dimensions x, y are independent and uniformly distributed in the interval $[0, 10]$. The payload dimension t is generated by sampling the standard normal distribution. The goal is to compute the median of payload values over the points contained in randomly generated spatial regions of varying size that covers from 10% up to 90% of the dataset domain. As shown in Fig. 13, SQLite and PostgreSQL employ different acceleration strategies but are unable to provide queries at interactive rates. As already stated, MonetDB lacks any sort of index to accelerate spatial queries, translating to a constant latency, no matter the number of filtered points. We highlight that all database solutions perform exact quantile computations, which makes necessary to scan all elements filtered by the spatial query. The regular build of QDS (with p-digest) was the only method able to provide quantile queries at interactive rates. QDS qnt computation time is dominated by merge operations. When measured individ-

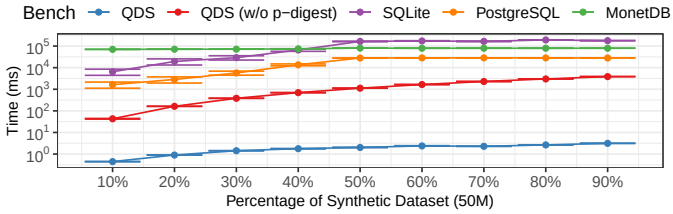


Figure 13. Comparison of $qnt(0.5)$ computation on a synthetic dataset using QDS, QDS without p-digest and database alternatives. As shown, QDS can provide quantile queries at interactive rates.

ually, merging times were less than one millisecond while using compression parameter $\delta = 50$.

Memory usage. Memory usage was a relevant factor when designing p-digest. The ability to share pivots and payload data, as well as memory saving strategies, prevent QDS size to be directly proportional to the p-digest *compression (δ)* parameter. While the average number of data items per p-digest is small, it is necessary to use a quantile sketch algorithm to enable quantile computation in large datasets, as shown in Fig. 13. As datasets become larger, allocating a buffer to store temporary data from the naïve approach becomes worse, since its size is proportional to the number of elements to compute the quantile. As observed in Table 2, the variation of *compression (δ)* parameter values has little impact on memory usage. Compression ratio ranges from $1.16x$ up to $1.34x$ when compared with the naïve solution.

9 Discussion

In this section, we further discuss issues related to QDS performance, applicability, and limitations. We first highlight that QDS improves on NC, Immens and Hashedcubes with respect to both capabilities, since these systems only support count queries and performance as described in Sec. 8. We also notice that having an (approximate) description of the distribution of a dataset is more powerful than using a parametric distribution, such as the Gaussians (used by GC). In fact, using p-digest we can retrieve the (approximate) values for moment statistics. However, the quantiles of a parametric distribution fitted to a dataset are in general far from the original ones (as illustrated in Fig. 8). This makes QDS widely applicable for the analysis of large spatiotemporal datasets. An interesting application scenario is the analysis of ensemble datasets in which different model predictions are put together to represent the diversity of the phenomenon under study.

A limitation of QDS compared to GC is the fact that it can only deal with univariate distributions (due to a limitation of p-digest) and, therefore, treats its payload dimensions as independent variables. Finally, while we have shown that QDS achieves a good approximation, it does not provide error bounds. We intend to investigate how to quantify and communicate the uncertainty in the approximation to the user. Also, we want to perform a formal user study to evaluate the use of QDS and the supported visualizations.

10 Conclusions and Future Work

In this paper, we presented QDS, a fast and memory efficient data structure that supports real-time (virtually immediate feedback) data exploration based on order statistics

Table 2
Compression results for different p-digest configurations.

dataset	p-digest compression							
	naïve		$\delta = 25$		$\delta = 50$		$\delta = 100$	
	memory	time	memory (compression)	time	memory (compression)	time	memory (compression)	time
flights	12.9 GB	05:35 m	9.9 GB (1.31 ×)	07:57 m	10.5 GB (1.22 ×)	08:09 m	10.9 GB (1.18 ×)	08:19 m
green-taxis	9.0 GB	01:54 m	6.7 GB (1.34 ×)	03:02 m	7.0 GB (1.30 ×)	03:05 m	7.2 GB (1.25 ×)	03:08 m
green-taxis-small	8.4 GB	01:52 m	6.7 GB (1.25x)	02:50 m	7.0 GB (1.20 ×)	02:54 m	7.2 GB (1.16 ×)	02:57 m
yellow-taxis-small	NA		12.7 GB (-)	54:57 m	12.9 GB (-)	55:59 m	13.3 GB (-)	56:04 m

on large multidimensional datasets. We believe that these capabilities open a large number of opportunities to design novel visual encodings and interaction techniques. In fact, other visualizations (matrix heatmaps, attributed network and etc) based on averages could be adapted to use their robust counterparts. Furthermore, we want to explore the possible use of QDS to speed-up computations in machine learning techniques for non-Gaussian distributions such as quantile regression and quantile based clustering. We see the coupling of cutting edge data sketching techniques with powerful precomputed indices to support interactive visual analytics as a promising future research direction. For example, we would like to explore how we can use of matrix and tensor sketching techniques to support the execution of complex analytical algorithms interactively. Another research direction is to define a data sketch to represent multivariate distributions with features similar to how t-digest can represent univariate distributions. To the best of our knowledge we are not aware of a solution to this problem. We believe the queries provided by QDS provide changes in the mindset during the analysis, allowing users to reason on the likelyhood of a hypothetical scenario like in Fig. 1. We intend to investigate these ideas in the future.

Acknowledgments

The authors wish to thank the anonymous reviewers for their valuable comments and suggestions. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq 308851/2015-3 and CNPq 140313/2017-6.

References

[1] Z. Liu and J. Heer, "The Effects of Interactive Latency on Exploratory Visual Analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2122–2131, 2014.

[2] L. Battle, R. Chang, and M. Stonebraker, "Dynamic Prefetching of Data Tiles for Interactive Visualization," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 1363–1375.

[3] Z. Liu, B. Jiang, and J. Heer, "imMens: Real-time Visual Querying of Big Data," in *Computer Graphics Forum*, vol. 32, no. 3pt4. Wiley Online Library, 2013, pp. 421–430.

[4] L. Lins, J. T. Klosowski, and C. Scheidegger, "Nanocubes for Real-Time Exploration of Spatiotemporal Datasets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2456–2465, Dec 2013.

[5] C. A. L. Pahins, S. A. Stephens, C. Scheidegger, and J. L. D. Comba, "Hashedcubes: Simple, Low Memory, Real-Time Visual Exploration of Big Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 671–680, Jan 2017.

[6] J. Matejka and G. Fitzmaurice, "Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics Through Simulated Annealing," in *Proc. Conference on Human Factors in Computing Systems (CHI)*. ACM, 2017, pp. 1290–1294.

[7] Z. Wang, N. Ferreira, Y. Wei, A. S. Bhaskar, and C. Scheidegger, "Gaussian Cubes: Real-Time Modeling for Visual Exploration of Large Multidimensional Datasets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 681–690, Jan 2017.

[8] T. Dunning and O. Ertl, "Computing Extremely Accurate Quantiles Using t-Digests," https://github.com/tdunning/t-digest, accessed: 2018-07-18.

[9] K. Potter, J. Kniss, R. Riesenfeld, and C. R. Johnson, "Visualizing Summary Statistics and Uncertainty," in *Computer Graphics Forum*, vol. 29, no. 3. Wiley Online Library, 2010, pp. 823–832.

[10] R. Maciejewski, A. Pattath, S. Ko, R. Hafen, W. S. Cleveland, and D. S. Ebert, "Automated box-cox transformations for improved visual encoding," *IEEE transactions on visualization and computer graphics*, vol. 19, no. 1, pp. 130–140, 2013.

[11] M. Correll and M. Gleicher, "Error bars Considered Harmful: Exploring Alternate Encodings for Mean and Error," *IEEE transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2142–2151, 2014.

[12] G. A. Rousselet, J. J. Foxe, and J. P. Bolam, "A Few Simple Steps to Improve the Description of Group Results in Neuroscience," *European Journal of Neuroscience*, vol. 44, no. 9, pp. 2647–2651, 2016.

[13] T. L. Weissgerber, N. M. Milic, S. J. Winham, and V. D. Garovic, "Beyond Bar and Line Graphs: Time for a New Data Presentation Paradigm," *PLoS Biology*, vol. 13, no. 4, p. e1002128, 2015.

[14] H. Wickham and L. Stryjewski, "40 Years of Boxplots," *Am. Statistician*, 2011.

[15] "Kick the bar chart habit," *Nature Methods*, vol. 11, pp. 113 EP –, 01 2014. [Online]. Available: https://doi.org/10.1038/nmeth.2837

[16] M. Kay, T. Kola, J. R. Hullman, and S. A. Munson, "When (ish) is my bus?: User-centered visualizations of uncertainty in everyday, mobile predictive systems," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 5092–5103.

[17] M. Fernandes, L. Walls, S. Munson, J. Hullman, and M. Kay, "Uncertainty Displays Using Quantile Dotplots or CDFs Improve Transit Decision-Making," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018, p. 144.

[18] D. Fisher, I. Popov, S. Drucker *et al.*, "Trust me, I'm Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster," in *Proc. Conference on Human Factors in Computing Systems (CHI)*. ACM, 2012, pp. 1673–1682.

[19] S. Chaudhuri, B. Ding, and S. Kandula, "Approximate Query Processing: No Silver Bullet," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 511–519.

[20] J. Jo, W. Kim, S. Yoo, B. Kim, and J. Seo, "Swifttuna: Responsive and incremental visual exploration of large-scale multidimensional data," in *Proc. Pacific Visualization Symposium (PacificVis)*, April 2017, pp. 131–140.

[21] D. Moritz and D. Fisher, "What Users Don't Expect about Exploratory Data Analysis on Approximate Query Processing Systems," in *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*. ACM, 2017, p. 9.

[22] N. Ferreira, D. Fisher, and A. C. Konig, "Sample-oriented Task-driven Visualizations: Allowing Users to Make Better, More Confident Decisions," in *Proc. Conference on Human Factors in Computing Systems (CHI)*. ACM, 2014, pp. 571–580.

[23] D. Moritz, D. Fisher, B. Ding, and C. Wang, "Trust, but Verify: Optimistic Visualizations of Approximate Queries for Exploring Big Data," in *Proc. Conference on Human Factors in Computing Systems (CHI)*. ACM, 2017, pp. 2904–2915.

[24] F. Miranda, L. Lins, J. Klosowski, and C. Silva, "TopKube: A Rank-Aware Data Cube for Real-Time Exploration of Spatiotemporal Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. PP, no. 99, pp. 1–1, 2017.

[25] J. Peng, D. Zhang, J. Wang, and J. Pei, "Aqp++: Connecting approximate query processing with aggregate precomputation for interactive analytics," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 1477–1492.

[26] H. Doraiswamy, N. Ferreira, T. Damoulas, J. Freire, and C. T. Silva, "Using Topological Analysis to Support Event-guided Exploration in Urban Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2634–2643, 2014.

[27] R. Maciejewski, S. Rudolph, R. Hafen, A. Abusalah, M. Yakout, M. Ouzzani, W. S. Cleveland, S. J. Grannis, and D. S. Ebert, "A Visual Analytics Approach to Understanding Spatiotemporal Hotspots," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 2, pp. 205–220, 2010.

[28] L. Wilkinson, "Visualizing Big Data Outliers Through Distributed Aggregation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 256–266, Jan 2018.

[29] J. S. Rosenthal, *A First Look at Rigorous Probability Theory*, 2nd ed. World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2006.

[30] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches," *Foundations and Trends in Databases*, vol. 4, no. 1–3, pp. 1–294, 2012.

[31] J. M. Phillips, "Coresets and sketches," *arXiv preprint arXiv:1601.00617*, 2016.

[32] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and Beyond: New Aggregation Techniques for Sensor Networks," in *Proc. International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004, pp. 239–249.

[33] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi, "Mergeable Summaries," *ACM Transactions on Database Systems*, vol. 38, no. 4, p. 26, 2013.

[34] Z. Karnin, K. Lang, and E. Liberty, "Optimal Quantile Approximation in Streams," in *Symp. on Foundations of Computer Science (FOCS)*, Oct 2016, pp. 71–78.

[35] D. Felber and R. Ostrovsky, "A randomized online quantile summary in $\mathcal{O}((1/\varepsilon)\log(1/\varepsilon))$ words," *Theory of Computing*, vol. 13, no. 14, pp. 1–17, 2017.

[36] Y. Ben-Haim and E. Tom-Tov, "A Streaming Parallel Decision Tree Algorithm," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 849–872, 2010.

[37] M. Greenwald and S. Khanna, "Space-efficient online computation of quantile summaries," *SIGMOD Records*, vol. 30, no. 2, pp. 58–66, May 2001.

[38] J. Kenney and E. Keeping, *Mathematics of Statistics*, ser. Mathematics of Statistics. Van Nostrand company, 1954, no. v. 1. [Online]. Available: https://tinyurl.com/ybpyupad

[39] R. Fraiman and G. Muniz, "Trimmed Means for Functional Data," *Test*, vol. 10, no. 2, pp. 419–440, 2001.

[40] Airlines for America, "U.S. Passenger Carrier Delay Costs," https://tinyurl.com/ycmxgcoy, accessed: 2018-07-18.

[41] US Department of Transportation, "On-Time Performance Dataset," https://tinyurl.com/y7tngze8, accessed: 2018-07-18.

[42] US Bureau of Transportation Statistics, "Air Travel Consumer Report," https://tinyurl.com/yde9nwo4, accessed: 2018-07-18.

[43] NYC Taxi and Limousine Commission, "Taxi Trip Records," https://tinyurl.com/q66cby3, accessed: 2018-07-18.

[44] NYPD, "NYC Police Department announces street closures and expected traffic delays for October 11-12th 2014," https://tinyurl.com/ybyooj4w, accessed: 2018-07-18.

[45] SpatiaLite, "SpatiaLite," https://tinyurl.com/d9re6ss, accessed: 2019-01-19.

[46] PostGIS, "Spatial and Geographic Objects for PostgreSQL," https://tinyurl.com/ycptpbsv, accessed: 2019-01-19.

[47] MonetDB B.V., "GeoSpatial | MonetDB," https://tinyurl.com/yal5gwev, accessed: 2019-01-19.

Cicero A. L. Pahins Cícero A. L. Pahins is a Ph.D candidate in Computer Science from Federal University of Rio Grande do Sul (UFRGS), where he conducts research activities in the area of Scalability, Exploration and Interactive Visualization of Big Data through the development of innovative data structures. He has experience in subjects like visualization and data analysis, scientific visualization, spatial data structures and graphic hardware.

Nivan Ferreira Nivan Ferreira is an Assistant Professor at Universidade Federal de Pernambuco (UFPE) in Brazil. He received a BSc in Computer Science and MSc in Mathematics from UFPE and PhD in Computer Science from New York University. Nivan was also a Post-Doc at the Department of Computer Science at the University of Arizona. Nivan's research focuses on many aspects of interactive data visualization, in particular systems and techniques for analysis spatiotemporal datasets.

João L. Comba João L. Comba is a Full Professor at the Instituto de Informática of the Universidade Federal do Rio Grande do Sul (UFRGS). Dr. Comba has a Ph.D. in Computer Science from Stanford University, a MSc in Systems Engineering and Computation from UFRJ (Brazil) and a BSc in Computer Science from UFRGS. His current research is on Visual Data Analysis, with emphasis on visual analytics, spatial data structures and high-performance computing. He is co-chair of the Visualization Corner column of the journal Computing in Science & Engineering.