Marviq: Quality-Aware Geospatial Visualization of Range-Selection Queries Using Materialization

Liming Dong*
Tsinghua University
dlm14@mails.tsinghua.edu.cn

Qiushi Bai
UC Irvine
qbai1@ics.uci.edu

Taewoo Kim
UC Irvine
taewok2@ics.uci.edu

Taiji Chen
UC Irvine
taijic@uci.edu

Weidong Liu
Tsinghua University
liuwd@tsinghua.edu.cn

Chen Li
UC Irvine
chenli@ics.uci.edu

Abstract

We study the problem of efficient spatial visualization on a large data set stored in a database using SQL queries with ad-hoc range conditions on numerical attributes, for example, a spatial scatterplot of taxi pickup events in New York between 1/1/2015 and 3/10/2015. We present a novel middleware-based technique called Marviq. It divides the selection-attribute domain into intervals, and precomputes and stores a visualization for each interval. These results are called MVS and stored as tables in the database. We can compute an exact visualization for a request by accessing MVS and retrieving additional records from the base table. To further reduce the latter time, we present algorithms for using MVS to compute an approximate visualization that satisfies a user-specified similarity threshold. We show a family of functions with certain properties that can use this technique. We present an improvement by dividing the MVS intervals into smaller intervals and materializing low-resolution visualization for these intervals. We report the results of an extensive evaluation of Marviq, including a user study, and show its high performance in both space and time.

CCS Concepts

• **Information systems → Middleware for databases**.

Keywords

Spatial data, Visualization, Quality guarantee, Marviq.

*This work was partially done during his visit to UCI.

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389730

ACM Reference Format:
Liming Dong, Qiushi Bai, Taewoo Kim, Taiji Chen, Weidong Liu, and Chen Li. 2020. Marviq: Quality-Aware Geospatial Visualization of Range-Selection Queries Using Materialization. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3318464.3389730

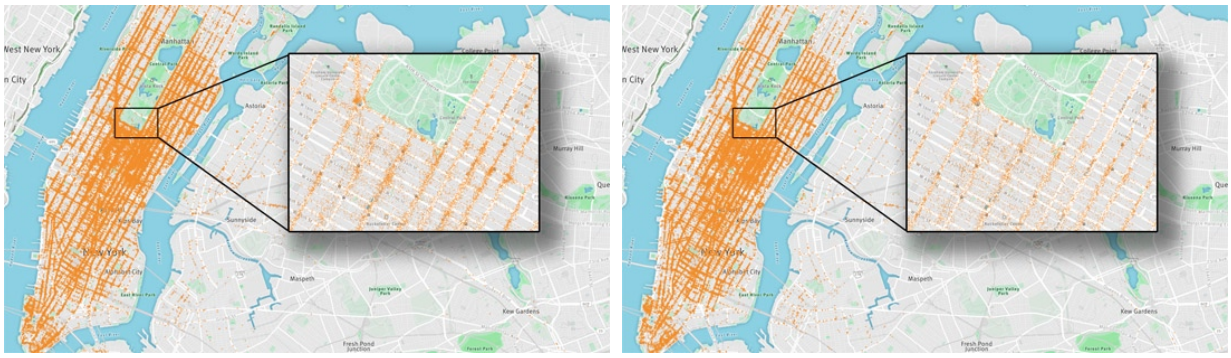
1 Introduction

Spatial data is widely used in a plethora of domains due to the increasing number of GPS-enabled devices and prevalent location-based services. Visualization of spatial data in these applications can help users easily gain insights from the data and make important decisions. Through an intuitive yet powerful interface, users can quickly see spatial data distributions, identify important relationships and properties of the entities, and observe valuable patterns and anomalies. As an example, consider a table of records about taxi pickup events in New York stored in a database. Each record includes the time, location, and other attributes of a pickup event. Suppose we want to visualize the pickup events of a time range (e.g., between 1/1/2015 and 3/10/2015) as a spatial scatterplot. The following is the SQL query:

```
SELECT location
FROM taxi
WHERE time BETWEEN '1/1/2015' AND '3/10/2015';
```

Figure 1(a) shows the scatterplot of the results, where each dot represents a pickup event. Managers of taxi and rideshare companies can compare the scatterplots of different time ranges to decide vehicle-scheduling strategies.

We study how to efficiently compute a spatial visualization from a large data set stored as a table. We focus on a type of spatial visualization with a range condition on an attribute of the table. An important requirement for *interactive* visualization is responsiveness, i.e., a user-facing request should be served efficiently, ideally in milliseconds [10, 28, 41]. Compared to common visualization types such as bar charts and line graphs, spatial visualization has two challenges: (**C1**) The amount of data to visualize can be much larger. It is



(a) Using data in [1/1/2015, 3/10/2015] (2 months + 10 days) (b) Using data in the first 2 months

Figure 1: Scatterplot of New York taxi pickup events for a time interval.

computationally expensive to retrieve these spatial points from the backend database, send them to the frontend over a network, and render them on the interface. (**C2**) Since the range selection in a request can be ad-hoc, it is not practical to precompute a visualization for every possible range.

In this paper we propose a novel approach called "Marviq," which stands for "materialization for spatial visualization with quality guarantee." Its main idea is to divide the domain of the selection attribute into multiple intervals. For each interval, we precompute its visualization and store it in a structure called "MVS," which is saved in the backend database as tables. For a visualization request with a range condition, we retrieve the materialized visualizations from MVS, as well as the spatial data in the residual "gap" range from the base table. Using these results we can compute an exact visualization. In the running example, suppose we compute and store a scatterplot for each month of taxi pickup events. For the request of Figure 1(a), we first retrieve the precomputed scatterplots of January and February 2015, and compute their union. We then retrieve those pickup events from 3/1/2015 to 3/10/2015, and combine them with the earlier scatterplot to have the final result. This method is efficient since the scatterplots of the first two months are already precomputed and materialized.

Our experiments show that the time of retrieving those additional records from the base table can still be a bottleneck. Thus we want to further reduce it, especially for applications requiring a high query throughput. To this end, we consider how to further improve the performance based on the observation that an approximate visualization using partial results can be acceptable by the user. For instance, Figure 1(b) is an approximate scatterplot using the data of the first two months, which already shows a very similar distribution, even in a zoomed-in region shown in the figure. Such an approximate visualization is often acceptable especially because nowadays devices of visualization interfaces such as desktops, laptops, or cell phones have a limited resolution.

When deciding how to compute an approximate visualization, we are facing an additional challenge: (**C3**) how to know if the visualization computed using partial results is "good enough" for the user? Thus we need to do reasoning about the quality of an approximate visualization. We show for a given quality function to compute the similarity between two visualizations, how to estimate the quality of an approximate visualization using MVS. We further improve MVS by dividing its intervals (e.g., months) into finer-granularity intervals (e.g., weeks), and materialize a low-resolution visualization for each small interval. We identify certain properties that need to be satisfied by this function in order to use Marviq. One advantage of Marviq is that it can leverage the storage, indexing, and querying capabilities of the underlying database, making it easy and efficient to implement. In summary, we make the following contributions:

- We present a technique called "Marviq" that can efficiently support spatial visualization for queries with an arbitrary range selection condition (Section 2).
- We show how to use Marviq to compute an approximate visualization using its MVS structure, and analyze its quality for a given similarity function. We present techniques for constructing a high-quality MVS efficiently (Section 3).
- We show a family of similarity functions supported by Marviq. We also generalize the solution to the case of multi-attribute conditions, heatmap visualization, and zoom in/out operations (Section 4).
- We improve the performance of MVS by dividing an interval to small intervals and precomputing a low-resolution visualization for each small interval. We study how to analyze the quality of results using the visualizations of these small intervals (Section 5).

- We report the results of an extensive evaluation of Marviq on real data sets, including a user study. Experiments show its low storage size and high query performance (response time in milliseconds) (Section 6).

1.1 Related Work

Visualization is a broad topic studied in many communities. Here we mainly focus on efficiency-related work. A survey [16] summarized studies on interactive data analytics and visualization, and there are several recent studies on this topic [21, 23, 25, 38].

Approximate Query Processing (AQP). There are many techniques for computing approximate answers to queries [2, 17, 26, 33, 36, 50, 51]. Sampling is one of the commonly used AQP methods. For instance, BlinkDB [2] uses stratified sampling to generate samples, and utilizes them to compute error-bounded results. AQP++ [37] integrates samples and precomputed aggregated results to answer queries. Other related studies include [4, 5, 12, 14, 18, 31, 35, 37, 39, 44]. For instance, Sample+Seek [12] combines data samples for high-selectivity predicates and an inverted index for low-selectivity predicates to answer aggregation queries. In our experiments, we chose Sample+Seek as a representative technique and compared it with Marviq. One advantage of Marviq is that it can be used to compute an exact result. Furthermore, it performs much better when computing an approximate visualization with a quality guarantee, as shown in the results in Section 6.

Datacube-based approaches. Related studies include [10, 11, 20, 22, 27, 29, 46]. Most of them do not focus on spatial visualization and do not support ad-hoc query conditions. ImMens [29] and Falcon [32] support fixed-bin ranges without supporting arbitrary ranges. Nanocubes [27] supports queries with arbitrary ranges by using in-memory cubes, but cannot support large data sets exceeding the memory size. For example, the authors reported that a dataset of 220 million records used about 45GB memory. Marviq uses a backend database to store its precomputed results, thus can handle much larger data sets (e.g., 1.3 billion records in the the New York taxi dataset managed using a low-end server).

Progressive visualization. There are solutions that show visualization results progressively [5, 9, 10, 14, 19, 31]. For instance, DICE [10] uses random and stratified samples and other techniques to present an approximate result at first, then incrementally updates the result. Pangloss [31] uses Sample+Seek [12] to do progressive visualization. Hillview [5] builds an incrementally updated spreadsheet for browsing a large dataset based on sketch [6]. Marviq can also be used to support progressive visualization, for example, by incrementally retrieving additional results from the base table.

Prefetching-based approaches. Example techniques are [3, 7, 40, 48]. For instance, ForeCache [3] divides visualizations into tiles and prefetches them based on predicted user behaviors. Atlas [7] uses predictive caching together with the backend database to support ad-hoc queries over time series datasets. IDEA [15] treats query results as variables, and reuses them in subsequent ad-hoc queries.

Visualization using big data systems. These techniques use Hadoop, Spark, Hive, Impela, etc. to compute visualizations [5, 8, 13, 41, 49]. For instance, HadoopViz [13] and GeoSparkViz [49] use Hadoop and Spark to generate high-resolution visualizations. Their focus is on offline construction, not on interactive visualization for queries with ad-hoc conditions. There are database solutions that use specialized hardware, such as GPU-based MapD/OmniSci [30].

Other solutions. VAS [35] computes a high-quality sample as an approximate visualization for scatterplots. It assumes results are already available (e.g., retrieved from a database) and focuses on generating good samples. Our focus is on reducing the time of retrieving query results from the backend database, and we experimentally compared these two approaches (Section 6). Kyrix [41] provides a model to create scalable visualizations and a gallery of visualizations. It does not support the aforementioned visualization with ad-hoc selection conditions on a numerical attribute when the domain cannot be modeled as multiple discrete categories.

2 Marviq Overview

System architecture: We consider a typical three-tier architecture (Figure 2), which consists of a backend database, a frontend visualization interface, and a middleware layer in between. Let T be a relational table in the database, where each record represents a spatial object such as a point of interest (POI), a taxi pickup event, or a tweet. The schema of the table includes a primary key called *Id*, a spatial attribute called *Point* represented as geo coordinates, and other attributes about each object. Table 1 shows an example data set of taxi pickup events, where the `PickupLoc` attribute stores the spatial location of an event, and the `PickupTime` attribute stores the time of the event. To see the pickup events between `1/1/2015` and `1/31/2015`, a user submits a query to the database to retrieve spatial objects in this time interval, and visualize the results on the frontend.

Visualization requests with selection conditions: A visualization request from the frontend retrieves records from the table using the following query:

```
SELECT location
FROM T
WHERE A BETWEEN a1 AND a2;
```

Table 1: Sample data of taxi pickup events

Id	PickupLoc	PickupTime	...
2014061721380019	(-74.0083, 40.7042)	6/17/2014 21:38:00	...
2014081414340025	(-73.9125, 40.6209)	8/14/2014 14:34:00	...
2014091217520078	(-73.9981, 40.7225)	9/12/2014 17:52:00	...

For simplicity we first consider the case where the query has a condition on a single attribute, and will generalize the results to the case of multi-attribute conditions. There are various ways to visualize spatial objects, such as scatterplots (where each record is shown as a dot) and heatmaps (where records are aggregated in a grid). We first focus on spatial scatterplot, then generalize the results to spatial heatmap. A scatterplot V is a mapping from a spatial pixel (x, y) to a 0/1 bit, where 1 means the geolocation corresponding to the pixel (x, y) has data. We use "$|V|$" to represent the number of 1-pixels in V. Notice (x, y) is a *logical* pixel, which may correspond to multiple physical pixels in the frontend display, e.g., it could correspond to 4×4 display pixels.

MVS: Precomputing Scatterplots for Intervals: Using Marviq, we partition the domain of the selection attribute A into multiple intervals, and store a precomputed visualization of the spatial objects in each interval I. This scatterplot is called the *exact visualization* (or "EV" for short) for interval I, denoted as "EV_I." In the current example, the attribute `PickupTime` spans from `1/1/2009` to `12/31/2015`. As shown in Figure 3, we first divide the time domain into intervals, e.g., month intervals, and compute a scatterplot for each month. The first scatterplot is for the data from `5/1/2009` to `5/31/2009`. These EV's are stored in a structure called MVS, which stands for *materialized visualization results*. This structure can be constructed either offline or maintained incrementally on the fly as more visualization requests are received. See Section 3.2 for details.

Consider a visualization request with a query range C. As shown in Figure 4, let C include intervals δ_1, I_1, I_2, ..., I_m, and δ_2, where $I_1, \ldots, I_m$ are the MVS intervals covered by C completely. Let their union be α. Intervals I_l and I_r are respectively the left and right intervals partially overlapping with C, if any. In addition, $\delta_1 = C \cap I_l$, $\delta_2 = C \cap I_r$. For each I_i $(i = 1 \ldots m)$, we retrieve the corresponding EV_{I_i} from the database, and compute their union $V_\alpha = \cup_{k=1}^{m} EV_{I_k}$. We also retrieve the records in the residual intervals δ_1 and δ_2. Using these records, together with V_α, we compute the final scatterplot for the user.

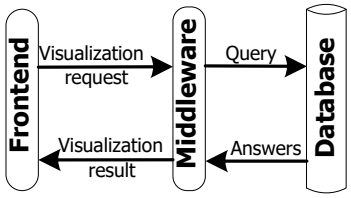


Figure 2: Visualization architecture for Marviq.

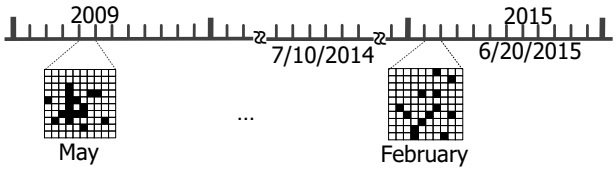


Figure 3: Marviq precomputes and stores a scatterplot EV_I for each interval I.

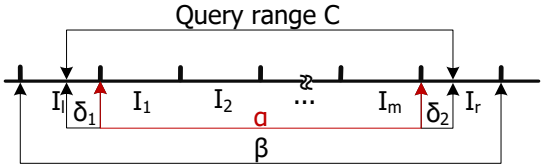


Figure 4: Computing a visualization using MVS.

3 Quality-Aware Approximate Visualization

A computational bottleneck in the aforementioned method is the step of retrieving records in the residual intervals δ_1 and δ_2. (See Section 6.3 for some experimental results.) We consider how to reduce this time by computing an approximate visualization that is "good enough" for the user. To quantify the notion of quality, we assume we are given a function $\mathcal{F}$ that measures the similarity between two spatial visualizations. Given two spatial visualizations V_1 and V_2, the function computes a value $\mathcal{F}(V_1, V_2)$ as their similarity. The quality function $\mathcal{F}$ depends on the type of visualization. For example, quality functions for scatterplot include perceptual hash [47], mean squared error [34], PSNR [34], and SSIM [45].

We assume each visualization request (with a query Q and a range C) has a similarity threshold τ. Any approximate visualization with a similarity above τ is acceptable to the user. Formally, let $Q(T)$ be the answers to Q, and $V(Q)$ be the exact visualization generated using $Q(T)$. We want to compute an approximate visualization V_a that meets the quality threshold, i.e.,

$$\mathcal{F}\big(V(Q), V_a\big) \geq \tau.$$

3.1 Jaccard-Based Visualization

We consider a commonly used similarity function, namely Jaccard, and show how to derive a bound on the quality of V_α for this function. Given two scatterplots V_1 and V_2, their interaction $V_1 \cap V_2$ is a scatterplot that maps a pixel (x, y)

to a 0/1 value that is a conjunction of $V_1(x, y)$ and $V_2(x, y)$. Their union $V_1 \cup V_2$ is defined similarly using a logical OR operation. The Jaccard similarity between two scatterplots V_1 and V_2 is defined as

$$\mathcal{J}(V_1, V_2) = \frac{|V_1 \cap V_2|}{|V_1 \cup V_2|}.$$

Two scatterplots are identical if and only if their similarity is 1, and they do not share any 1-bit pixels if and only if their similarity is 0.

As shown in Figure 4, consider $\beta = I_l \cup I_1 \cup \ldots \cup I_m \cup I_r$, and $V_\beta = V_\alpha \cup EV_{I_l} \cup EV_{I_r}$. The following lemma gives a lower bound on the similarity between V_α and $V(Q)$.

Lemma 3.1. *For a query as shown in Figure 4, we have:*

$$\mathcal{J}(V_C, V_\alpha) \geq \mathcal{J}(V_\beta, V_\alpha). \quad (1)$$

Based on this lemma, we use MVS to compute an approximate scatterplot that meets the quality requirement as follows. We first access MVS to compute V_α and V_β, and estimate the quality based on Lemma 3.1. If the quality meets the requirement τ, we return V_α to the user. Otherwise, we access the table T to retrieve the records in the ranges δ_1 and δ_2, and compute their visualization V. We return $V_\alpha \cup V$ as the final, exact scatterplot to the frontend. Alternatively, we can query the table T to retrieve the records in δ_1 and compute its scatterplot EV_{δ_1}, if the quality of $EV_{\delta_1} \cup V_\alpha$ meets the requirement, we can return this approximate scatterplot. Otherwise, we need to retrieve the records in δ_2. Optionally we can access the records in δ_2 first.

3.2 Constructing a High-Quality MVS

In this subsection we discuss how to efficiently construct a high-quality MVS. We first consider the case of having a query workload, then study the case where queries are received on the fly, and we construct and maintain MVS adaptively. A main focus in both cases is how to decide and modify the intervals.

3.2.1 MVS Construction for a Query Workload Consider a given workload $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ and a storage budget $\mathbb{B}$ for the MVS. For simplicity, we first consider the case where each EV is stored as a bitmap, then all the EV's have the same storage size. The problem becomes finding a set of intervals $|\mathcal{M}| \leq \mathbb{B}$ that can maximize the overall efficiency of the queries in the workload. The main idea of our algorithm is to first use all the start and end points of query intervals to construct the initial structure, then reduce the number of intervals. For this purpose, we introduce a loss value for removing a dividing point X_i as:

$$loss(X_i) = N(X_{i-1}, X_i) \cdot L(X_i) + N(X_i, X_{i+1}) \cdot R(X_i). \quad (2)$$

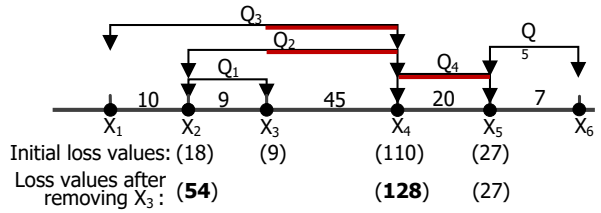


Figure 5: Loss values of dividing points.

The loss consists of two values, one for left queries and one for right queries. Take the left value as an example. $N(X_{i-1}, X_i)$ is the number of spatial objects in the interval $X_{i-1}X_i$, and $L(X_i)$ is the number of queries on the left of X_i that cover interval $X_{i-1}X_i$ and end at X_i. The intuition behind this value is that once X_i is removed, the EV of $X_{i-1}X_i$ can no longer be used to compute a partial visualization of these queries, and the number of additional records that need to be retrieved is $N(X_{i-1}, X_i)$. Similarly, $R(X_i)$ is the number of queries on the right of X_i that start from X_i and cover interval X_iX_{i+1}, and $N(X_i, X_{i+1})$ is the number of objects in this interval. Figure 5 shows a few intervals and their corresponding number of records. If we remove point X_4, then $EV_{X_3X_4}$ can no longer be used to compute the partial visualization for the red segments of Q_2 and Q_3, with X_3X_4 (in red) as the interval from which additional records need to be retrieved. Similarly, $EV_{X_4X_5}$ can no longer be used to compute the visualization for the red interval of Q_4. In the figure, initially we have $loss(X_4) = 45 * 2 + 20 * 1 = 110$. After removing X_3, we have $loss(X_4) = 54 * 2 + 20 * 1 = 128$. We can use a greedy algorithm to iteratively choose a point with a minimum loss value to remove, and merge its two adjacent intervals until the total storage is within the given space.

Our discussion so far assumes all EV's have the same size. If EV's have different sizes, we can treat the storage size of each EV as its cost, and compute the size reduction of removing X_i as

$$size\big(EV(X_{i-1}X_i)\big) + size\big(EV(X_iX_{i+1})\big) - size\big(EV(X_{i-1}X_{i+1})\big).$$

We use the ratio of loss/(size reduction) as a criterion to choose the next point with the minimum ratio to remove.

3.2.2 Adaptive Construction of MVS In the case where a query workload is not given, we need an online method that can adaptively construct and update the MVS structure. We develop such a technique as follows. At the beginning, we divide the domain of the attribute A to fixed-length intervals, and construct an initial MVS. As more queries arrive, we maintain a history of recent queries, denoted as $\mathcal{Q}$, which presumably represent the distribution of future queries. Based on these queries, we maintain statistics for the points and intervals. Periodically we merge some of the intervals by removing points to decrease the total storage, or split some of them with a higher storage cost.

Point loss and removal: For each point, similar to the previous case, we compute its loss value based on the queries in the recent history Q. To merge intervals to reduce the storage space, we choose a point with the minimum loss, and merge its two adjacent intervals. As before, we also need to update the loss values of the start and end points of the new interval. When a new query Q with a range $[a, b]$ is added to Q, we modify the loss values of two adjacent points. Similarly, when an old query is removed from the history Q, we need to modify the loss values of the two affected points.

Interval benefit and splitting: In order to decide which interval to split, we keep a benefit value for each interval, which is the number of start and end points of the queries in Q that fall into this interval. Intuitively, the larger the benefit value is for the interval, the more queries the MVS structure can answer (possibly partially) by splitting this interval. We perdiodically choose an interval with the largest benefit to split, e.g., into two equal-size intervals. Similarly, we can incrementally update these benefit values when a new query is added to Q and an old query is removed from Q.

Notice that we do not have to update the point-loss values and interval-benefit values for each new query. As for histogram-like data structures, we can update these values periodically for new queries, while the content in the MVS structure is still accurate.

MVS Storage and Maintenance. One advantage of MVS is that it can be efficiently stored and accessed as tables in the backend database. The EV of a parent interval can be stored as a list of coordinates of 1-pixels, or as a bitmap if the number of 1-pixels is very large. For the former method, we can use two tables to store the MVS structure, one for the start and end points of intervals, and one for coordinates of the pixels. In the presence of data updates (insertions and deletions), MVS can be maintained incrementally and efficiently. When a new record t is inserted, we set the value of the geo-position in the corresponding EV to 1. In the case of deleting a record, we need to update the value of its geo-position in its EV based on whether there is still a remaining record at this position in the EV.

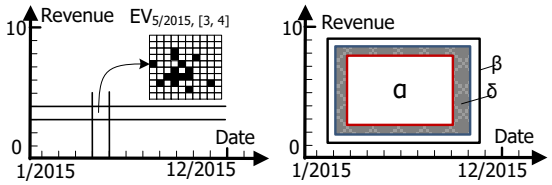
4 Generalizing Marviq

Our Marviq results so far are developed for scatterplot visualization and Jaccard function when a query condition is on a single numerical attribute. In this section, we extend the results to general cases by relaxing these assumptions.

4.1 Multi-attribute Conditions

To support queries with multiple selection conditions, we can extend the one-dimensional MVS structure to a multi-dimensional structure. The main idea is that instead of using intervals, we use multi-dimensional regions, and keep statistics within each region. The MVS structure, methods for estimating quality of an approximate visualization using the structure, as well as the algorithms for generating an approximate visulization, can all be extended accordingly.

To illustrate the generalization, we use an example of two attributes and show how to construct and utilize MVS. Consider a table of spatial objects with two numerical attributes, namely `Date` and `Revenue`. Figure 6(a) shows an MVS structure for these two attributes. We divide the entire domains of these two attributes into two-dimensional regions. That is, the x-axis has `Date` intervals and the y-axis has `Revenue` intervals. We generate an exact visualization (EV) for each region that consists of a `Date` interval and a `Revenue` interval. For example, $EV_{5/2015,[3,4]}$ is the EV of the records with a `Date` value in May 2015 and a `Revenue` value in $[3, 4]$.



(a) MVS **(b) Quality estimation**

Figure 6: MVS for two attributes.

For a visualization request $U(Q, \tau)$ with a condition C on these two attributes, we can use the EV's of the regions covered by C to compute an approximate visualization V_a. To estimate the quality of V_a, we can extend Lemma 3.1 to this two-dimensional case. In particular, as shown in Figure 6(b), α is the union of maximal number of MVS regions that are covered by C, δ is the area $C - \alpha$, and β is the union of minimal number of MVS regions that cover C. The following inequality shows a lower bound of the quality of EV_α:

$$\mathcal{J}(V(Q), V_\alpha) \geq \mathcal{J}(V_\beta, V_\alpha),$$

which is an extension of Lemma 1.

As the number of selection attributes increases, the number of regions (i.e., "cells" in the data cube) can be large. In this case, the number of spatial objects satisfying many selection conditions could be much smaller, and they can be efficiently retrieved without using MVS, possibly using a multi-attribute composite index. We can use data warehouse techniques to construct MVS by judiciously choosing some of the cells to compute and store an EV (e.g., [20]). Since the intervals in a query can cover multiple MVS intervals on each attribute, we can answer this request by sending a disjunctive query to the original table to retrieve those records in the non-materialized cells, sending another disjunctive query to MVS to retrieve the EV's for those materialized cells, and computing a visualization.

4.2 General Quality Functions

We extend the Jaccard-based results in Section 3 to a family of similarity functions that satisfy two properties.

Subset Increasing Monotonicity. As described in Section 3.1, if the scatterplot V_α for the contained intervals does not satisfy the user threshold τ, we can retrieve more records, e.g., those in interval δ_1. This method assumes that retrieving more data points in the query range C increases the visualization quality. In other words, it means adding more data points satisfying the query condition can increase the quality. This property is also needed by the improved MVS technique presented in Section 5. Here is a formal definition of the property. A similarity function $\mathcal{F}$ is said to be *subset increasing monotonic* if for any three intervals I^1, I^2, and I^3 of the condition attribute A, if $I^1 \subseteq I^2 \subseteq I^3$, then $\mathcal{F}(V_{I^1}, I_{I^3}) \leq \mathcal{F}(V_{I^2}, V_{I^3})$.

Superset Decreasing Monotonicity. Lemma 3.1 shows how to use the similarity between V_α and V_β as a lower bound of the similarity between V_α and V_C. The property means adding more data points in the range $\beta - C$ reduces its similarity to V_α. This property is formally defined as follows. A similarity function $\mathcal{F}$ is said to be *superset decreasing monotonic* if for any three intervals I^3, I^4, and I^5 of the condition attribute A, where $I^3 \subseteq I^4 \subseteq I^5$, we have $\mathcal{F}(V_{I^4}, V_{I^3}) \geq \mathcal{F}(V_{I^5}, V_{I^3})$.

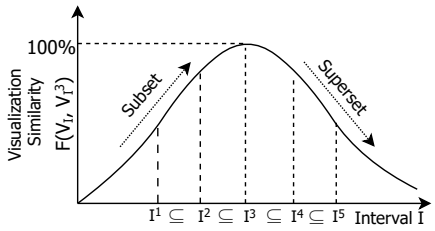


Figure 7: Monotonic properties of a quality function

These two properties are illustrated in Figure 7. Intuitively, the first property means that adding more data points (in the interval I^3) to a subset of points in I^3 will make the scatterplot more similar to V_{I^3}. The second property means that adding more data points to a superset of points in I^3 will make the scaterplot less similar to V_{I^3}. These two properties are satisfied by many similarity functions, such as mean squared error (MSE), PSNR [34], and SSIM [45], and the Marviq results in Section 3 are applicable to these functions. For instance, the MSE between two $m \times n$ scatterplots A and B is defined as:

$$MSE(A, B) = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} (A(i, j) - B(i, j))^2.$$

We can show that MSE satisfies both monotonicity properties. For instance, the following is an extension of Lemma 1 based on Figure 4:

$$MSE(V_C, V_\alpha) \leq MSE(V_\beta, V_\alpha). \quad (3)$$

4.3 Heatmap Visualization

Next we illustrate how to extend the results based on scatterplot to heatmap. A heatmap is a spatial grid with a density value for each cell, which corresponds to the number of spatial objects in the cell. A heatmap can be used to generate an image with a smoothing step, by taking an average of the neighbor-cell numbers for each cell. Heatmap can be viewed as a generalization of scatterplot by allowing each cell to have a count value, not just a 0/1 value.

The main idea is to keep the density for each cell when constructing MVS, and use the density to derive a lower bound of the quality. For instance, Figure 8 shows the generation of EV's for heatmap, in which we set the resolution of an EV to be the granularity of the heatmap. For each pixel of an EV, we keep the number of records in this pixel. For two consecutive intervals and a pixel, the count value for the pixel in their EV is the *summation* of the corresponding counts for the pixel in the two separate EV's. There are various quality functions for heatmap such as MSE.

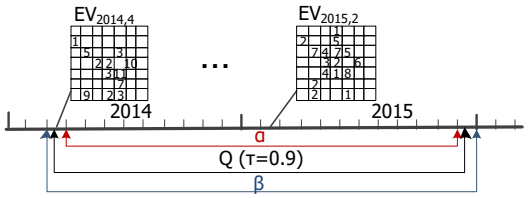


Figure 8: MVS for heatmap.

Figure 8 shows an example query with a range condition $C = [3/10/2014, 12/15/2015]$ and a quality requirement $\tau = 0.9$. For the intervals in $[4/2014, 11/2015]$ that are completely covered by C, we can use the summation of the counts in their EV's to generate an approximate heatmap V_α. Similarly, let V_β be the corresponding approximate heatmap using the EV's of the intervals in $[3/2014, 12/2015]$. Inequality 3 still holds for heatmap and MSE. Based on this result, we can use $MSE(V_\beta, V_\alpha)$ as an upper bound of the error distance. If V_α cannot meet the requirement τ, we retrieve the records in $[3/10/2014, 3/31/2014]$ or $[12/1/2015, 12/15/2015]$ or all these records to compute a heatmap with a higher quality. Other results extend correspondingly.

4.4 Supporting Zooming and Panning

So far we assume the visualization has a fixed location and resolution (e.g., the entire US map). Very often a user wants to zoom in, zoom out, and pan on the map. To support these operations, we can divide the spatial space into multiple levels, and build an MVS structure for each level. As the user zooms into a smaller area, the resolution of the result visualization increases, causing the size of the structure to increase. We can divide the MVS structure at this

level into multiple tiles, and only retrieve the tiles visible in the current map area [3]. As the user pans on the map, we can correspondingly retrieve the related tiles. Notice that we do not need to build an MVS structure for each level for two reasons. First, we can use the tiles of a high resolution to compute a tile with a low resolution. Second, for a level with a high resolution (e.g., at the local district level), the number of records satisfying both the spatial condition and the selection condition in the query tends to be much smaller. By using indexes such as B+ tree and R-tree, we can efficiently retrieve these records and show all of them on the map. Given a smaller number of records, we can efficiently compute an exact visualization.

5 Improving MVS Using Low-Resolution Visualizations

In MVS, increasing the number of intervals improves the quality the visualization using a union of its EV's, as well as the efficiency of the generated query. Meanwhile, the storage overhead increases as well. In case the quality of the results using the precomputed EV's does not meet the query requirement, we need to retrieve all the data in the remainder interval in the query condition, since we do not have any information about the data distribution within the interval. In this section, we present an improved structure called MVS$^+$, analyze how to use it to estimate the quality of a visualization, and develop an algorithm for using this structure to compute an approximate visualization. Again, we focus on scatterplot and Jaccard similarity, and the results can be generalized to other quality functions and heatmap.

5.1 Adding Low-Resolution Visualizations

The main idea of MVS$^+$ is the following. We divide the domain of the attribute $\mathcal{A}$ into intervals of two granularities, namely *parent intervals* and *child intervals*, where each parent interval consists of multiple child intervals. We precompute visualizations as follows. (1) Store an EV for each parent interval. (2) For each child interval c, compute the EV of the records from the beginning of its parent interval until (including) c, and store a *low-resolution visualization* (or "LV" for short) for c. Given an exact visualization with a resolution of $N \times N$, a low-resolution visualization is a grid of size $k \times k$, where k is a tunable parameter. Each cell in the grid stores the number of 1-bit pixels in the corresponding $N/k \times N/k$ region in the exact visualization.

Figure 9 shows an MVS$^+$ structure for the taxi data, using years and months as two interval granularities. The structure has an EV for each year, denoted as $EV_{2009} \ldots EV_{2015}$, with a default resolution of 10×10. Each year is divided to 12 month intervals. Consider October 2015 (i.e., 10/2015) as an example. We compute the EV of the records of the year 2015 up to this month, i.e., from 1/2015 to 10/2015, denoted as $BEV_{10/2015}$, where B stands for "before." To save space, instead of storing this EV, we want to store a visualization with a low resolution. We divide the spatial region into a 2×2 grid (i.e., $k = 2$), where each cell corresponds to a 5×5 region in the original resolution. Each cell stores the number of 1-bit pixels in the corresponding region. For instance, the top-left cell has a value 7, since the corresponding cell has 7 pixels of value 1. This low-resolution structure is denoted as $BLV_{10/2015}$. In general, we use "BEV_c" to represent the EV for the records up to a child interval c in the same parent interval, and use BLV_c to denote the corresponding low-resolution visualization. Similarly, $AEV_{10/2015}$ is the EV of those months after this month in the same year, and we use ALV to denote the corresponding low-resolution visualization.

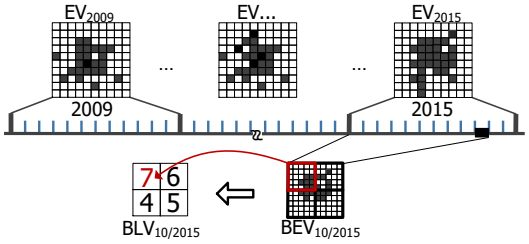


Figure 9: MVS$^+$ structure for taxi data.

5.2 Tighter Quality Bound of EV Results

Next we discuss how to use MVS$^+$ to answer a visualization request. We focus on the case where the query range C starts at the beginning of a parent interval, and ends at the end of a child interval. The results can be generalized to the case where the condition starts and ends at an arbitrary value.

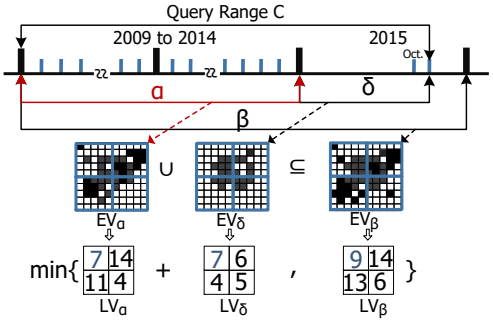


Figure 10: Maximal value of $|EV_{[1/2009,10/2015]}|$.

Consider the query in Figure 10, whose range is [1/2009, 10/2015]. Let α denote the range [1/2009, 12/2014], so $EV_\alpha = EV_{2009} \cup \ldots \cup EV_{2014}$. One way to estimate the quality of EV_α is to use the right-hand side of Inequality 1. Its denominator in the bound is based on the pessimistic assumption that the 1-bit pixels in year 2015 are all in the query range C. We can derive a tighter bound by using the additional information

in the monthly LV's in 2015 stored in the structure. Recall that the Jaccard similarity between EV_α and $V(Q)$ is:

$$\mathcal{J}(V(Q), EV_\alpha) = \frac{|EV_\alpha|}{|EV_\alpha \cup BEV_{10/2015}|}. \quad (4)$$

For EV_α, we use LV_α to denote its corresponding 2×2 visualization. For each cell c_i in the grid, the maximal value of the denominator is when EV_α and $BEV_{10/2015}(c_i)$ have disjoint sets of 1-bit pixels in this cell c_i. Therefore, the denominator in the function is no greater than

$$\sum_{i=1}^{4}\big(LV_\alpha(c_i) + BLV_{10/2015}(c_i)\big).$$

The number of 1-bit pixels in the query range C should be no greater than $EV_{[2009,2015]}$. Let $\beta = [2009, 2015]$ and $\delta = [1/2015, 10/2015]$, thus $LV_\delta = BLV_{10/2015}$. Let $n = 4$ denote the number of cells in each LV. We have:

$$\mathcal{J}(V(Q), EV_\alpha) \geq \frac{\sum_{i=1}^{n} LV_\alpha(c_i)}{\sum_{i=1}^{n} min\{LV_\alpha(c_i) + LV_\delta(c_i), LV_\beta(c_i)\}}. \quad (5)$$

Notice that LV_δ is $BLV_{10/2015}$ of the last month in the query range C. The above result can be generalized to as follows.

Lemma 5.1. *Inequality 5 is valid for a general query Q with a range C that starts at the beginning of a parent interval and ends at the end of a child interval. In the inequality, α is the union of the parent intervals completely covered by C, δ is the overlap between C and the last parent interval that partially overlaps with C, β is the union of the parent intervals that overlap with C, and n is the number of cells in each low-resolution visualization.*

5.3 Tighter Bound Using Child Intervals

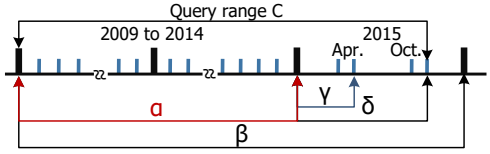


Figure 11: Estimating the quality of $EV_{[1/2009,4/2015]}$.

In the running example, if $EV_{[2009,2014]}$ cannot meet the quality requirement τ based on the lower bound in Lemma 5.1, we want to include results in some of the months in year 2015 to compute an approximate visualization with a higher quality. In order to decide the months of which we retrieve records, we estimate its quality based on the LV's of months in 2015. For instance, we want to estimate the quality of the visualization of data in $\alpha \cup \gamma = [1/2009, 4/2015]$. As illustrated in Figure 11, we have

$$EV_{\alpha\cup\gamma} = EV_\alpha \cup BEV_{4/2015} = EV_\alpha \cup EV_\gamma,$$

and

$$EV_{\alpha\cup\delta} = EV_\alpha \cup BEV_{10/2015} = EV_\alpha \cup EV_\delta.$$

Recall that

$$\mathcal{J}(V(Q), EV_{\alpha\cup\gamma}) = \frac{|EV_\alpha \cup BEV_{4/2015}|}{|EV_\alpha \cup BEV_{10/2015}|} = \frac{|EV_\alpha \cup EV_\gamma|}{|EV_\alpha \cup EV_\delta|}.$$

The following inequality shows a lower bound of the quality of $EV_{\alpha\cup\gamma}$, which is an extension of Inequality 5.

$$\mathcal{J}(V(Q), EV_{\alpha\cup\gamma}) \geq \frac{\sum_{i=1}^{n} max\{LV_\alpha(c_i), LV_\gamma(c_i)\}}{\sum_{i=1}^{n} min\{max\{LV_\alpha(c_i), LV_\gamma(c_i)\} + (LV_\delta(c_i) - LV_\gamma(c_i)), LV_\beta(c_i)\}}. \quad (6)$$

Similar to Inequality 5, LV_γ and LV_δ are the BLV of the last interval of γ and δ, respectively. The above result can be generalized as the following lemma.

Lemma 5.2. *Inequality 6 is valid for a general query. Those symbols are defined in Lemma 5.1, and γ is a range of child intervals in the last parent interval overlapping with C.*

5.4 Answering Requests Using MVS$^+$

Algorithm 1 shows how to compute a visualization using MVS$^+$. It extends the earlier MVS-based algorithm by utilizing the LV's of child intervals to obtain a tighter bound of the quality of an approximate visualization based on the data from these intervals. Different strategies can be used in Line 5 to extend the range of V_α to include additional child intervals γ to meet the τ requirement. One strategy is to estimate the quality before sending the residual query. We extend γ by one child interval each time and use Lemma 5.2 to calculate the quality bound ℓ of $EV_{\alpha\cup\gamma}$ until we find a γ that meets τ, and then send the residual query. Another strategy is progressively sending a residual query and estimating the quality. That is, we send a query for the extended γ and use Lemma 5.1 to calculate the quality of $EV_{\alpha'}$, where $\alpha' = \alpha \cup \gamma$, and repeat the process until we get an $EV_{\alpha'}$ that meets τ.

MVS$^+$ storage and incremental maintenance. In the full version [24], we show that MVS$^+$ can be efficiently stored as tables in the database, and queried to return relevant data for visualization. It can also be incrementally maintained efficiently in the presence of data updates. We also report the experimental results in [24].

Algorithm 1: Visualization using MVS+

Input: A visualization request $U(Q, \tau)$ with a range condition C on the numerical attribute $\mathcal{A}$;
Output: A visualization with a quality at least τ.

Let $I_1, \ldots, I_m$ be the parent intervals completely covered by C;
$V_\alpha = EV_{I_1} \cup \ldots \cup EV_{I_m}$;
Use Lemma 5.1 to calculate a quality bound ℓ of EV_α;
while $\ell < \tau$ **do**
    Increase the number of child intervals in γ in the last parent interval partially overlapping with C;
    Calculate a quality bound ℓ of $EV_{\alpha \cup \gamma}$;
end
Retrieve data in residual intervals γ;
Use the retrieved data to compute a visualization V;
return $EV_\alpha \cup V$;

6 Experiments

In this section we report experimental results of Marviq, including a user study, to evaluate visualizations produced by different techniques. We used three real datasets, as described in Table 2. The first dataset, denoted as Foursquare [1], contained Foursquare check-in geo-locations and times all over the world from April 2012 to September 2013. The second dataset, Tweet, included about 100 million tweets in the U.S. from November 2015 to June 2017 collected using the Twitter's public API [42]. The third dataset, Taxi [43], was the records of New York City Yellow Cab and Green Taxi trips from January 2009 to June 2016. Each record contained information about a taxi trip, including its pick-up and drop-off geo-locations, times, trip distance, payment method, and fare. For each dataset, we kept three attributes, including the ID, event time, and geo-location.

Table 2: Datasets.

Dataset	Record # (millions)	Size (GB)
Foursquare	33	2.1
Tweet	100	18.0
Taxi	1,300	320.0

We used PostgreSQL 9.6 as the database with the following configuration: *shared memory* was 8GB, *work_mem* was 256MB, and *maintenance_work_mem* was 2GB. We ran the database on a server running CentOS 7 with an Intel Xeon E3 CPU, 64GB RAM, 1 GB Ethernet NIC, and a 1TB SSD drive. For each dataset, a visualization request had a range condition on its time attribute with a B+ tree. We implemented Marviq in a middleware layer written in Python v2.7 on the same machine as the database.

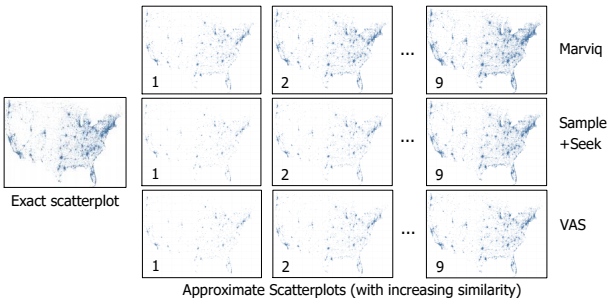
6.1 User Study

We conducted a user study for two purposes: (1) to evaluate the quality of a visualization result computed by different techniques; and (2) to decide a similarity threshold for a quality function given to Marviq. We considered Sample+Seek [12, 31] and VAS [35] as two representative, state-of-the-art sampling-based techniques. Sample+Seek supports ad-hoc conditions, and uses a quality function called *distribution precision* to measure the similarity between two distributions. VAS does not support ad-hoc selection conditions and does not provide similarity-based quality guarantee.

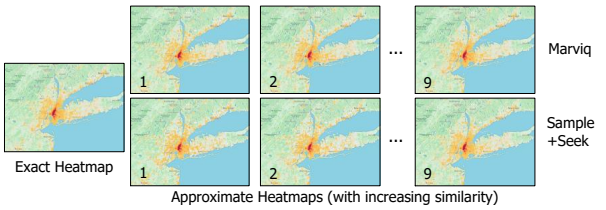
Setup. We invited 36 users to participate in the study. They were 11 females and 25 males, with an age between 20 and 30. For the Tweet dataset, we selected four geographic areas, namely the contiguous United States, Los Angeles, New York, and Chicago. We considered scatterplot and heatmap as two commonly used spatial visualization types. For each region and visualization type, we generated an exact visualization for the data of two time ranges, namely a week and a month. For each time range, we used Marviq to generate 9 approximate visualizations using 9 subintervals of the same length. Thus the generated visualizations had an increasing similarity to the exact one. For Sample+Seek and VAS, we also generated 9 approximate visualizations using 9 samples with an increasing size, and these visualizations had an increasing similarity too. Because VAS only supports scatterplot, we did not use it to generate heatmap. In total, we generated 376 images, including 16 exact visualizations (2 time rages * 4 regions * 2 visualization types), 216 approximate scatterplots in 24 groups (3 methods * 2 time rages * 4 regions), and 144 approximate heatmaps in 16 groups (2 methods * 2 time rages * 4 regions). Figure 12 shows example visualizations of two geographic areas.

For each of four regions, for each of the two time ranges, for each of the two visualization types, and for each of the three techniques, we asked each user to look at the exact visualization and 9 approximate ones, sorted in an increasing order of their similarity, without telling the user which method was used. Each approximate result had a degree between 1 and 9, while the least similar one had a degree of 1. We asked the user to specify the first visualization that they thought was similar enough to the exact result. We kept track of the degree of the "first good-enough" visualization selected by the user. There was also an option for the user if they did not see any "good enough" result. Each user spent around 32–47 minutes to do the evaluation. In total we collected 1,440 degree values (36 participants * 40 group approximate visualizations).

Results. Table 3(a) shows the results for scatterplots. For Sample+Seek and VAS, there were 44 and 19 cases where a



(a) Spatial scatterplot of the contiguous United States (US).



(b) Spatial heatmap of the New York (NY) region.

Figure 12: Example visualizations generated in the user study using 1 month of tweets.

user did not find an acceptable approximate result, respectively, while the number for Marviq was only 1. The average degree of acceptable results was 8.6, 6.4, and 5.4 for Sample+Seek, VAS, and Marviq, respectively. Table 3(b) shows the results of heatmaps. There was no case of unacceptable heatmap for Marviq, and the number for Sample+Seek was 10. The average degree of acceptable results was 2.2 and 5.0 for Marviq and Sample+Seek, respectively. In terms of the number of acceptable visualizations, users preferred visualizations generated by Marviq than those produced by Sample+Seek. An interesting observation was that for Sample+Seek, a spatial point in an approximate scatterplot with a lower degree could disappear in one with a higher degree. This behavior made the results less preferred by some users. For Marviq, the scatterplot of a larger degree was always a superset of one with a smaller degree. We also found that the method of computing the ϵ parameter in VAS was not suitable in our settings, and we had to adjust it manually to produce high-quality scatterplots.

We computed the Jaccard quality of results produced by Marviq and VAS, whereas Sample+Seek used its proposed distribution precision function. (The Jaccard quality of scatterplots with a distribution precision of 0.02 was about 70%.) While they used different quality functions as well as different methods to generate the approximate visualizations, the user study results helped us decide comparable degrees of acceptable visualizations by these techniques, which was used in the performance comparison reported in Section 6.4.

Table 3: User study results. Each cell has values "a/b/c" for Marviq ("a"), Sample+Seek ("b"), and VAS ("c"), respectively.

(a) Scatterplot

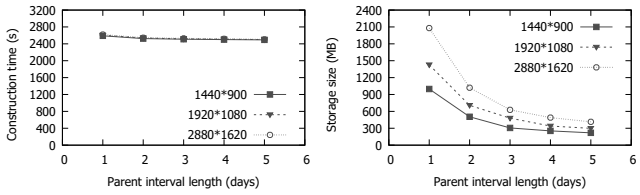
	US	NY	LA	Chicago
# of acceptable results	71/65/55	72/56/70	72/54/72	72/69/72
# of unacceptable results	1/7/17	0/16/2	0/18/0	0/3/0
Average degree of acceptable results	5.5/8.2/9.0	5.1/8.8/6.0	6.1/9.0/5.2	4.9/8.3/5.5
Average quality of acceptable results	65%/0.02/ 73%	57%/0.018/ 60%	70%/0.015/ 65%	53%/0.02/ 47%

(b) Heatmap

	US	NY	LA	Chicago
# of acceptable results	72/72/-	72/72/-	72/62/-	72/72/-
# of unacceptable results	0/0/-	0/0/-	0/10/-	0/0/-
Average degree of acceptable results	2.0/3.9/-	1.8/3.2/-	2.8/9.0/-	2.0/4.0/-
Average quality of acceptable results	30%/0.06/-	25%/0.03/-	37%/0.015/-	30%/0.06/-

6.2 Construction Time and Storage Space

We used the Taxi dataset to evaluate the time and space to construct and store MVS. We varied the parent interval length from 1 day to 5 days, and varied the visualization resolution from $1440 * 900$ to $2880 * 1620$, and each point (EV cell) occupied 4*4 pixels. The results are shown in Figure 13.



(a) Construction time (very similar for all resolutions) **(b) Storage size**

Figure 13: Construction and storage of MVS.

For each parent interval, the computation of its EV consisted of three steps, including retrieving its query results from the database, computing the EV, and writing the EV into the database. Experiments showed that 99% of the total time was spent in the first step. As we increased the length of each parental interval, the computation of each EV took longer time, and the total construction time was stable, indicating that most of the time was spent on retrieving the data from the disk inside the database and sending the data to the middleware. As shown in Figure 13(b), the size decreased as the parent-interval length increased, since more records

mapped to the same pixel. The size of MVS was relatively very small compared to the original data size. Also the MVS size increased sublinearly as the visualization resolution increased. The results for the other two data sets were similar, and were omitted due to the limited space. In the following experiments, we set the visualization resolution to be $1920 * 1080$, a resolution for many displays.

To evaluate the performance of different construction strategies, we used two months of taxi data. We constructed MVS$^+$ with 10 parent intervals with a length of 1 day and 10 child intervals for each parent interval. We used the total size of this structure as the storage budget. We created a workload of $1,000$ queries with $\tau = 0.9$, and each of them started within the second and third parent intervals and ended within the fourth and fifth parent intervals, with a length varied between 0.5 and 2 parent intervals. We first implemented a naive method, called FixedIntervals, which divided the attribute domain to 10 parent intervals evenly without considering any query workload. We then implemented the two construction strategies described in Section 3.2. One, called Offline, used the input query workload to construct MVS$^+$ within the storage budget. The other one, called Online, adjusted the structures adaptively based on the recent 100 queries. Table 4 shows the performance of these strategies, including the number of records in the constructed MVS$^+$, the construction time, and the percentage of queries that could be answered by using EV's only without retrieving additional records from the original table. Not surprisingly, the Offline strategy performed the best given its knowledge about the workload. The Online strategy was much better than FixedIntervals by adaptively adjusting its intervals based on recent queries.

Table 4: Performance of MVS$^+$-construction strategies.

	FixedIntervals	Offline	Online
MVS$^+$ size (record #)	184K	82.8K	110K
Mean response time (s)	0.95	0.34	0.77
% of queries answerable using EV's	48.4%	88.9%	81.6%

6.3 Visualization Performance

We created a workload of visualization requests as follows. We used the parent-interval size as a length unit to generate query intervals. We varied the query length from 1 to 5 parent intervals, and for each of them, we randomly generated $1,000$ queries with that length.

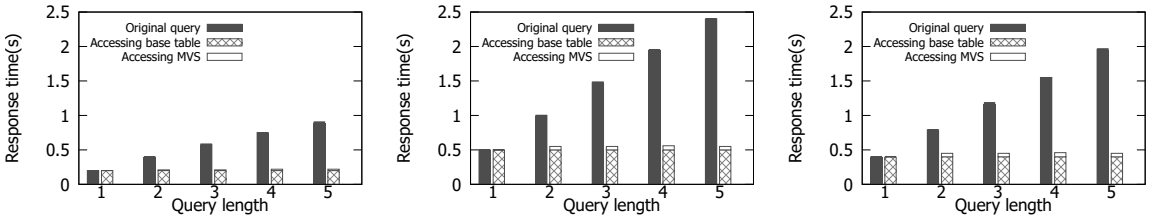
Time of computing an exact visualization result. We evaluated the performance of generating an exact result for a visualization request as described in Section 2. Figure 14 shows the response times of queries with different lengths on the three datasets. We can see that the runng time of the original query increased linearly as its length increased. However, the running time of using MVS in Marviq was very stable, which varied between $0.2s$ to $0.5s$ for the three data sets. The time included the time of retrieving MVS data of the contained intervals (I_α in Figure 3) and the time of retrieving records of the residual intervals (I_{β_1} and I_{β_2}) from the raw data table, which was the main bottleneck. This bottleneck is a motivation of computing approximate visualizations to reduce this time to have a higher query throughput and increase the responsiveness of the system, which is needed for applications with many concurrent visualization requests.

Comparing MVS and MVS$^+$. To have a fair comparison between MVS and MVS$^+$, we wanted to make MVS$^+$ have the same size as MVS. Three factors affect the MVS$^+$ size, including the parent interval size, the child interval number within a parent interval, and the LV resolution. Experiments showed that the parent interval size had the greatest impact on the performance. Therefore, we fixed the child interval number within a parent interval to be 10 and LV resolution to be 32*18, and we used the following ratio to control the MVS$^+$ size :

$$m = \frac{\text{parent interval length of MVS}^+}{\text{parent interval length of MVS}}.$$

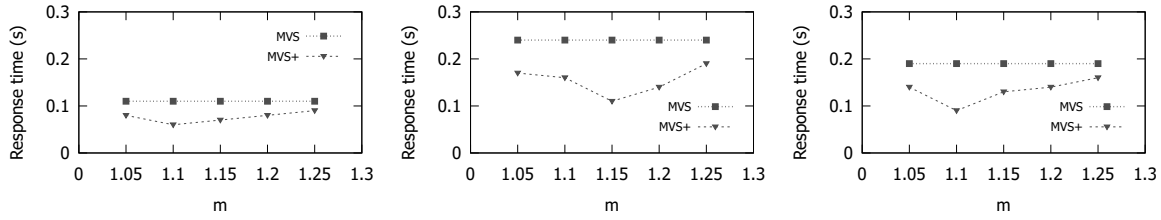
For each data set, we first constructed an MVS. We varied the ratio value m for an MVS$^+$ to achieve a similar storage size by adjusting the cell resolution in each LV. Each visualization request used $\tau = 70\%$ as its quality requirement. The average response time of these requests is shown in Figure 15. We can see that the MVS$^+$ structure always achieved a lower response time than MVS, and the reduction ratio varied from 5% to 54%. As m increased, the number of parent intervals in MVS$^+$ is smaller than MVS, thus LVs can have higher resolution. The response time using MVS$^+$ first decreased because higher LV resolution gave better bound estimation, but then increased because too long parent intervals of MVS$^+$ reduced the possibility of queries benefit from the EVs. There was an optimal m value for each dataset that can achieve the best performance. Since MVS$^+$ demonstrated superiority than MVS, in the remaining experiments, we mainly focused on this structure.

Time saving using MVS$^+$. We can use MVS$^+$ to reduce the response time further when users can accept approximate visualizations. For each parent interval length, we ran $1,000$ queries and measured their average response time. We used an optimal m ratio value to maximize the performance for each data set. Figure 16 shows the results for different quality values $\tau = 80\%$, 70%, and 60%. We can see that the response time decreased as the query length increased. The reason is that as the query length increased, more EV's can be used to generate the approximate visualization, and less data needed to be retrieved from the data table. These EV's can be retrieved from the database very efficiently. For instance, on



(a) Foursquare (b) Tweet (c) Taxi

Figure 14: Response times of computing an exact scatterplot using MVS.



(a) Foursquare (b) Tweet (c) Taxi

Figure 15: Response time of computing an approximate result using MVS and MVS$^+$ of different m ratios ($\tau = 70\%$).

the Taxi dataset, by using MVS$^+$ with a size less than 0.5% of the size of 1.3 billion records, Marviq computed an exact result using 0.4s, and an approximate visualization of quality $\tau = 70\%$ in less than 0.1s.

6.4 More Comparison with Sample+Seek

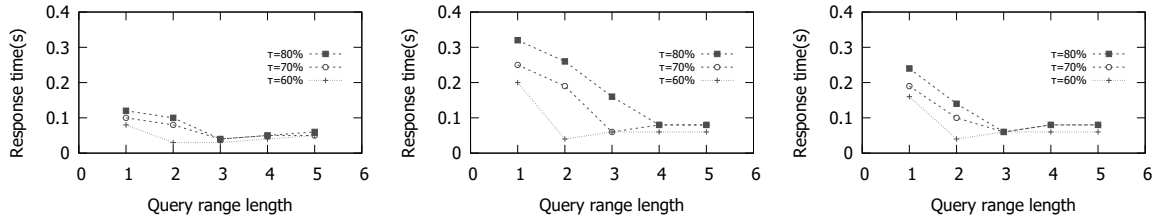
Since a heatmap can be viewed as a distribution of the grid cells, Sample+Seek can used to compute an approximate heatmap. We implemented Sample+Seek on a heatmap (with a resolution of 100*100) using its proposed distribution precision function. It precomputed a sample based on a given distance ϵ (corresponding to our $1 - \tau$) between the original and approximate visualizations. We used the twitter data set, varied ϵ from 0.100 to 0.025, and used a confidence 0.95. For each ϵ, we created the corresponding sample. The ϵ, sample size, and sampled record number are shown in Table 5. Since we needed to apply a selection condition on the sample before a group-by, it was more efficient to store it in the database than storing it in memory, so that we can utilize an index on the selection attribute to accelerate visualization requests. To give Sample+Seek its best setting, we stored its sample in the database.

Table 5: Using Sample+Seek on tweets.

Distance ϵ (our $1 - \tau$)	0.025	0.050	0.075	0.100
Sample size (MB)	550	137	62	34
Record # (millions)	16	4	1.78	1

We constructed an MVS$^+$ structure using 1,000 parent intervals, and each interval was divided into 10 child intervals, with a total size of 129MB. We created a workload of queries with a length ranged from 1 to 10 parent intervals, and there were 100 queries for each length. Given a distance ϵ, Sample+Seek accessed the corresponding sample table to compute an approximate visualization. Marviq treated ϵ as the quality requirement $1 - \tau$, used the EV's in MVS$^+$ to compute an initial approximation, and used a bound to decide whether to retrieve more data. The experiments showed that the bound derived using our method was loose, causing Marviq to retrieve all the records in the residual child interval and produce an *exact* visualization. In other words, in this use case, the main benefit of Marviq was the precomputed results in the EV's, and the produced result had a distance $\epsilon = 0$ (no error). Figure 17 shows the response time for different query lengths. The response time was stable for Marviq. As ϵ increased, the time of Sample+Seek decreased and the sample size also decreased. Marviq's time was around 0.48s, with a perfect visualization ($\epsilon = 0$). A main difference is that Marviq produced an exact visualization with comparable index size and response time, while Sample+Seek generated an approximate visualization with a probabilistic quality.

In Sample+Seek, the minimum number of records in the sample to get an ϵ-approximate visualization is $O(1/\epsilon^2)$. This number does *not* depend on the number of groups, which is very critical in spatial visualization. Therefore, distribution precision has its limitations to be used as a quality function for spatial visualization. Figure 18 shows the response time of computing an approximate scatterplot in the user study. Recall that the average degree of acceptable scatterplots of Sample+seek was 8.6, and the corresponding sample size was 1.5GB. Sample+Seek took about 3.5 seconds to return a result. In contrast, the average degree of acceptable scatterplots of Marviq was 5.4, and it can return a result in less than 0.2s



(a) Foursquare (m=1.1) (b) Tweet (m=1.15) (c) Taxi (m=1.1)

Figure 16: Response time of computing an approximate scatterplot using MVS$^+$.

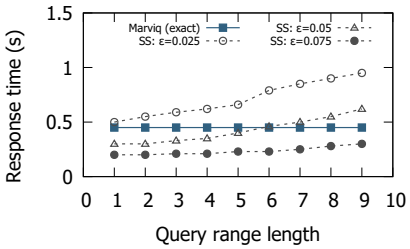


Figure 17: Response time of Marviq for an exact result and Sample+Seek ("SS") for an approximate result with a probabilistic quality.

with an MVS of 129MB. The comparison results of heatmap were similar to Figure 18, and omitted due to limited space.

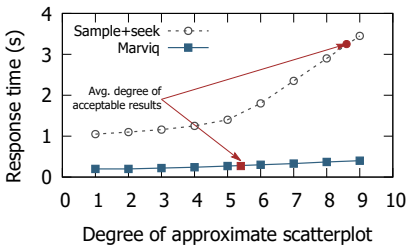


Figure 18: Response time of approximate scatterplots (1 week of tweets).

6.5 More Comparison with VAS

VAS [35] is a technique that relies on sampling to generate an approximate spatial visualization. Notice that VAS and Marviq have different settings. VAS focused on sampling without any pre-computation. Marviq assumes a predefined quality function and relies on pre-computed results to accelerate real-time visualization requests. We did an experiment with the purpose of evaluating the tradeoffs between these two approaches in terms of their storage size and running time for the case of a given quality function. In the experiment, we randomly generated queries of a length ranged from 1 to 3 parent intervals. For each query, we used all its query results as the original data, constructed 3 VAS samples with a ratio varied from 5% to 15%. For each sample, we computed the Jaccard similarity between its scatterplot and the one using all the results. We then formulated a corresponding visualization request using the original query and this quality as the τ requirement, and used MVS$^+$ to compute an approximate visualization that met the quality requirement. The results in Table 6 show that for the same quality requirement, Marviq significantly reduced the query response time (from more than 1,000 seconds to less than 1s.)

Table 6: VAS versus Marviq (using MVS$^+$).

VAS sample ratio	5%	10%	15%
VAS sample quality	85.5%	92.9%	94.8%
VAS sampling time	1,323.3s	1,469.9s	2,018.7s
Marviq response time	0.21s	0.24s	0.30s
MVS+ size ratio to original data	0.5%	0.5%	0.5%

We also report the experimental results of other quality functions in [24].

7 Conclusions

In this paper we studied the problem of efficient spatial visualization of a large data set stored in a database using SQL queries with ad-hoc range conditions on numerical attributes. We presented a novel middleware-based technique called Marviq. It divides the selection-attribute domain into intervals, and precomputes and stores a visualization in a structure called MVS. We show how to use MVS to compute an exact visualization or an approximate result with a quality guarantee based on a similarity function and a user-specified threshold. We showed a family of functions with certain properties that can use this technique. We presented an improvement by dividing the MVS intervals into smaller intervals and materializing low-resolution visualization for these intervals. We reported the results of an extensive evaluation of Marviq, including a user study, and show its high performance in both space and time.

8 Acknowledgments

We want to thank the anonymous reviewers for their constructive comments, Sadeem Alsudais at UC Irvine for her insightful comments, and Boling Ding at Alibaba Group for helpful discussions.

References

[1] Foursquare dataset, 2018. https://enterprise.foursquare.com/products/places.

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013*, pages 29–42, 2013.

[3] 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, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1363–1375, 2016.

[4] E. Bertini and G. Santucci. Give chance a chance- modeling density to enhance scatter plot quality through random data sampling. *Information Visualization*, 5(2):95–110, 2006.

[5] M. Budiu, P. Gopalan, L. Suresh, U. Wieder, H. Kruiger, and M. K. Aguilera. Hillview: A trillion-cell spreadsheet for big data. *PVLDB*, 12(11):1442–1457, 2019.

[6] M. Budiu, R. Isaacs, D. Murray, G. D. Plotkin, P. Barham, S. Al-Kiswany, Y. Boshmaf, Q. Luo, and A. Andoni. Interacting with large distributed datasets using sketch. In E. Gobbetti and W. Bethel, editors, *EGPGV16: Eurographics Symposium on Parallel Graphics and Visualization, Groningen, The Netherlands, June 6-10, 2016*, pages 31–43. Eurographics Association, 2016.

[7] S. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology, IEEE VAST 2008, Columbus, Ohio, USA, 19-24 October 2008*, pages 59–66. IEEE Computer Society, 2008.

[8] D. Cheng, P. Schretlen, N. Kronenfeld, N. Bozowsky, and W. Wright. Tile based visual analytics for twitter big data exploratory analysis. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, pages 2–4, 2013.

[9] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: Interactive analytics through pen and touch. *PVLDB*, 8(12):2024–2027, 2015.

[10] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. The case for interactive data exploration accelerators (ideas). In C. Binnig, A. Fekete, and A. Nandi, editors, *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, page 11. ACM, 2016.

[11] C. A. de Lara Pahins, S. A. Stephens, C. Scheidegger, and J. L. D. Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE Trans. Vis. Comput. Graph.*, 23(1):671–680, 2017.

[12] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample + seek: Approximating aggregates with distribution precision guarantee. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 679–694, 2016.

[13] A. Eldawy, M. F. Mokbel, and C. Jonathan. Hadoopviz: A mapreduce framework for extensible visualization of big spatial data. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 601–612, 2016.

[14] D. Fisher, I. O. Popov, S. M. Drucker, and m. c. schraefel. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*, pages 1673–1682, 2012.

[15] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting reuse for approximate query processing. *PVLDB*, 10(10):1142–1153, 2017.

[16] P. Godfrey, J. Gryz, and P. Lasek. Interactive visualization of large data sets. *IEEE Trans. Knowl. Data Eng.*, 28(8):2142–2157, 2016.

[17] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen. Approxhadoop: Bringing approximations to mapreduce frameworks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015*, pages 383–397, 2015.

[18] T. Guo, K. Feng, G. Cong, and Z. Bao. Efficient selection of geospatial data on maps for interactive and visualized exploration. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 567–582, 2018.

[19] J. Im, F. G. Villegas, and M. J. McGuffin. Visreduce: Fast and responsive incremental information visualization of large datasets. In X. Hu, T. Y. Lin, V. V. Raghavan, B. W. Wah, R. A. Baeza-Yates, G. C. Fox, C. Shahabi, M. Smith, Q. Yang, R. Ghani, W. Fan, R. Lempel, and R. Nambiar, editors, *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, pages 25–32. IEEE, 2013.

[20] Jia Yu and M. Sarwat. Accelerating spatial data visualization dashboards via a materialized sampling approach. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2020.

[21] L. Jiang, P. Rahman, and A. Nandi. Evaluating interactive data systems: Workloads, metrics, and guidelines. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1637–1644, 2018.

[22] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and interactive cube exploration. In I. F. Cruz, E. Ferrari, Y. Tao, E. Bertino, and G. Trajcevski, editors, *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 472–483. IEEE Computer Society, 2014.

[23] T. Kraska. Northstar: An interactive data science system. *PVLDB*, 11(12):2150–2164, 2018.

[24] L. Dong, Q. Bai, T. Kim, T. Chen, W. Liu and C. Li. Marviq: Quality-Aware Geospatial Visualization of Range-Selection Queries Using Materialization (Full Version). UC Irvine Technical Report, 2020.

[25] D. J. L. Lee and A. G. Parameswaran. The case for a visual discovery assistant: A holistic solution for accelerating visual data exploration. *IEEE Data Eng. Bull.*, 41(3):3–14, 2018.

[26] K. Li and G. Li. Approximate query processing: What is new and where to go? - A survey on approximate query processing. *Data Science and Engineering*, 3(4):379–397, 2018.

[27] L. D. Lins, J. T. Klosowski, and C. E. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2456–2465, 2013.

[28] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2122–2131, 2014.

[29] Z. Liu, B. Jiang, and J. Heer. *imMens*: Real-time visual querying of big data. *Comput. Graph. Forum*, 32(3):421–430, 2013.

[30] MapD demo. https://www.mapd.com/demos/taxis.

[31] D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017.*, pages 2904–2915, 2017.

[32] D. Moritz, B. Howe, and J. Heer. Falcon: Balancing interactive latency and resolution sensitivity for scalable linked visualizations. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, page 694, 2019.

[33] B. Mozafari, J. Ramnarayan, S. Menon, Y. Mahajan, S. Chakraborty, H. Bhanawat, and K. Bachhav. Snappydata: A unified cluster for streaming, transactions and interactice analytics. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017.

[34] T. N. Pappas, R. J. Safranek, and J. Chen. Perceptual criteria for image quality evaluation. *Handbook of image and video processing*, pages 669–684, 2000.

[35] Y. Park, M. J. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 755–766, 2016.

[36] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: Universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1461–1476, 2018.

[37] 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, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1477–1492, 2018.

[38] F. Psallidas and E. Wu. Provenance for interactive visualizations. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2018, Houston, TX, USA, June 10, 2018*, pages 9:1–9:8, 2018.

[39] S. Rahman, M. Aliakbarpour, H. Kong, E. Blais, K. Karahalios, A. G. Parameswaran, and R. Rubinfeld. I've seen "enough": Incrementally improving visualizations to support rapid decision making. *PVLDB*, 10(11):1262–1273, 2017.

[40] E. A. Rundensteiner, M. O. Ward, Z. Xie, Q. Cui, C. V. Wad, D. Yang, and S. Huang. Xmdvtool^{9}: : quality-aware interactive data exploration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 1109–1112, 2007.

[41] W. Tao, X. Liu, Y. Wang, L. Battle, Ç. Demiralp, R. Chang, and M. Stonebraker. Kyrix: Interactive pan/zoom visualizations at scale. *Comput. Graph. Forum*, 38(3):529–540, 2019.

[42] 2018. https://developer.twitter.com/en.html.

[43] 2018. https://github.com/fivethirtyeight/uber-tlc-foil-response.

[44] L. Wang, R. Christensen, F. Li, and K. Yi. Spatial online sampling and aggregation. *PVLDB*, 9(3):84–95, 2015.

[45] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[46] 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 Trans. Vis. Comput. Graph.*, 23(1):681–690, 2017.

[47] L. Weng and B. Preneel. A secure perceptual hash algorithm for image content authentication. In B. De Decker, J. Lapon, V. Naessens, and A. Uhl, editors, *Communications and Multimedia Security*, pages 108–121, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[48] J. Yu, R. Moraffah, and M. Sarwat. Hippo in action: Scalable indexing of a billion new york city taxi trips and beyond. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*, pages 1413–1414. IEEE Computer Society, 2017.

[49] J. Yu, Z. Zhang, and M. Sarwat. Geosparkviz: a scalable geospatial data visualization framework in the apache spark ecosystem. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management, SSDBM 2018, Bozen-Bolzano, Italy, July 09-11, 2018*, pages 15:1–15:12, 2018.

[50] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-OLA: generalized on-line aggregation for interactive analysis on big data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 913–918, 2015.

[51] X. Zhang, J. Wang, J. Yin, and S. Ji. Sapprox: Enabling efficient and accurate approximations on sub-datasets with distribution-aware online sampling. *PVLDB*, 10(3):109–120, 2016.