Visualization-Aware Sampling for Very Large Databases

Yongjoo Park, Michael Cafarella, Barzan Mozafari
University of Michigan, Ann Arbor, USA
pyongjoo@umich.edu, michjc@umich.edu, mozafari@umich.edu

Abstract—**Interactive visualizations are crucial in *ad hoc* data exploration and analysis. However, with the growing number of massive datasets, generating visualizations in interactive timescales is increasingly challenging. One approach for improving the speed of the visualization tool is via *data reduction* in order to reduce the computational overhead, but at a potential cost in visualization accuracy. Common data reduction techniques, such as uniform and stratified sampling, do not exploit the fact that the sampled tuples will be transformed into a visualization for human consumption.**

We propose a *visualization-aware* sampling (VAS) that guarantees high quality visualizations with a small subset of the entire dataset. We validate our method when applied to scatter and map plots for three common visualization goals: regression, density estimation, and clustering. The key to our sampling method's success is in choosing a set of tuples that minimizes a visualization-inspired loss function. While existing sampling approaches minimize the error of aggregation queries, we focus on a loss function that maximizes the visual fidelity of scatter plots. Our user study confirms that our proposed loss function correlates strongly with user success in using the resulting visualizations. Our experiments show that (i) VAS improves user's success by up to 35% in various visualization tasks, and (ii) VAS can achieve a required visualization quality up to $400\times$ faster.

I. Introduction

Data scientists frequently rely on visualizations for analyzing data and gleaning insight. For productive data exploration, analysts should be able to produce *ad hoc* visualizations in interactive time (a well-established goal in the visualization and human-computer interaction (HCI) community [1]–[10]). However, with the rise of big data and the growing number of databases with millions or even billions of records, generating even simple visualizations can take a considerable amount of time. For example, as reported in Figure 2, we found that the industry standard Tableau visualization system takes over 4 minutes on a high-end server to generate a scatterplot for a 50M-tuple dataset that is already resident in memory. (see Section VI-A for experimental details.) On the other hand, HCI researchers have found that visualizations must be generated in 500ms to 2 seconds in order for users to stay engaged and view the system as interactive [11]–[13]. Unfortunately, dataset sizes are already growing faster than Moore's Law [14] (the rate at which our hardware is speculated to improve), so technology trends will likely exacerbate rather than alleviate the problem.

This paper addresses the problem of interactive visualization in the case of *scatterplots* and *map plots*. Scatterplots are a well-known visualization technique that represent database records using dots in a 2D coordinate system. For example, an

(a) Stratified Sampling (overview)　　(b) Stratified Sampling (zoom-in)
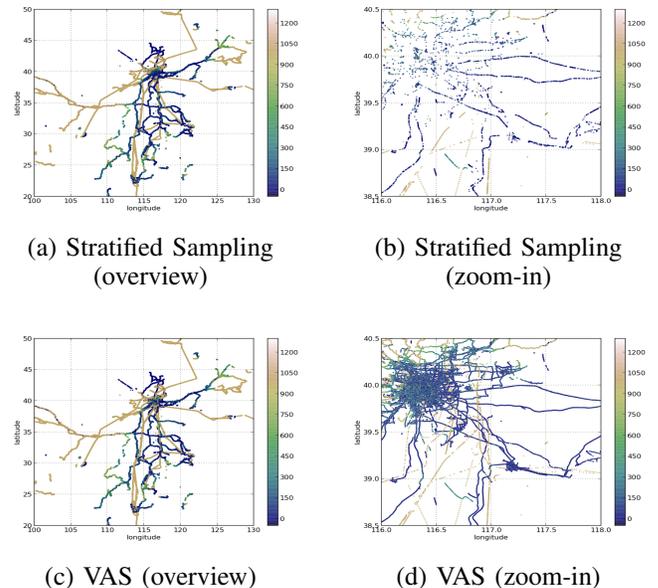
(c) VAS (overview)　　(d) VAS (zoom-in)

Fig. 1: Samples generated by fined-grained stratified sampling and our approach respectively. When the entire range is visualized, both methods seem to offer the visualization of the same quality. However, when zoomed-in views were requested, only our approach retained important structures of the database.

engineer may investigate the relationship between time-of-day and server-latency by processing a database of Web server logs, setting time-of-day as the scatterplot's X-axis and the server-latency as its Y-axis. Map plots display geographically-tied values on a 2D plane. Figure 1(a) is an example of a map plot, visualizing a GPS dataset from OpenStreetMap project with 2B data points, each consisting of a latitude, longitude, and altitude triplet (altitude encoded with color).

One approach for reducing the time involved in visualization production is via *data reduction* [15]. Reducing the dataset reduces the amount of work for the visualization system (by reducing the CPU, I/O, and rendering times) but at a potential cost to the quality of output visualization. Effective data reduction will shrink the dataset as much as possible while still producing an output that preserves all important information of the original dataset. Sampling is a popular database method for reducing the amount of data to be processed, often in the context of approximate query processing [16]–[24]. While uniform (random) sampling and stratified sampling are two
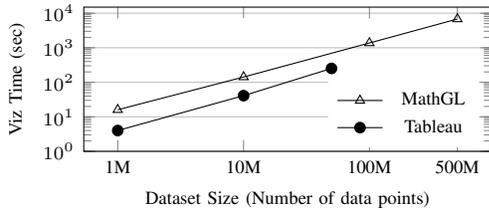
Fig. 2: The latency for generating scatter plot visualizations using Tableau and MathGL (a library for scientific graphics).

Fig. 3: Standard model of user interaction with the combined visualization and database system.

of the most common and effective approaches in approximate query processing [25], they are not well-suited for generating scatter and map plots: they can both fail to capture important features of the data if they are sparsely represented [4].

Figure 1 depicts an example using the Geolife dataset [26]. This dataset contains GPS-recorded locations visited by the people living in and around Beijing. In this example, we visualized 100K datapoints using both stratified sampling and our approach. For stratified sampling, we created a 316-by-316 grid and set the strata sizes (the number of datapoints in each cell) as balanced as possible across the cells created by the grid. In the zoomed-out overview plots, the visualization quality of the two competing methods seem nearly identical; however, when a zoomed-in plot is generated, one can observe that our proposed method delivers significantly richer information.

Previous Approaches — Architecturally, our system is similar to ScalaR [15], which interposes a data reduction layer between the visualization tool and a database backend; however, that project uses simple uniform random sampling. Researchers have attempted a number of approaches for improving visualization time, including binned aggregation [3]–[5], parallel rendering [9], [10], and incremental visualization [7], [8]. These methods are orthogonal to the one we propose here.

Our Goals — This paper tackles the technical challenge of creating a sampling strategy that will yield *useful and high-quality* scatter and map plots at *arbitrary zooming resolutions* with as *few sampled tuples* as possible. Figure 1(d) shows the plot generated by our proposed method, which we call Visualization-Aware Sampling (VAS). Using the same number of tuples as random and stratified sampling, VAS yields a much higher-fidelity result. The use of VAS can be specified as part of the queries submitted by visualization tools to the database. Using VAS, the database returns an approximate query answer within a specified time bound using one of multiple pre-generated samples. VAS chooses an appropriate sample size by converting the specified time bound into the number of tuples that can likely be processed within that time bound. VAS is successful because it samples data points according to a visualization-oriented metric that correlates well with user success across a range of scatter and map plot tasks.

Contributions — We make the following contributions:

- We define the notion of VAS as an optimization problem (Section III).
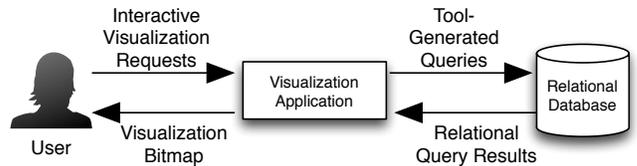- We prove that the VAS problem is NP-hard and an offer efficient approximation algorithm. We establish a worst-

case guarantee for our approximate solution (Section IV).
- In a user study, we show that our VAS is highly correlated with the user's success rate in various visualization tasks. We also evaluate the efficiency and effectiveness of our approximation algorithm over several datasets. We show that VAS can deliver a visualization that has equal quality with competing approaches, but using up to $400\times$ fewer data points. Alternatively, if VAS can process an equal number of data points as competing methods, it can deliver a visualization with a significantly higher quality (Section VI).

Finally, we cover related work in Section VII and conclude with a discussion of future work in Section VIII.

II. SYSTEM OVERVIEW

A. Software Architecture Model

Figure 3 shows the software architecture model that we focus on in this paper. This is a standard architecture supported by the popular Tableau system [27]. It is also similar to ScalaR's "dynamic reduction" software architecture [15]. The user interacts with a visualization tool to describe a desired visualization — say, a scatterplot of Web server time vs latency. This tool has been configured to access a dedicated RDBMS, and the schema information from the RDBMS is visible in the user's interaction with the tool. For example, in Tableau, a user can partially specify a desired scatterplot by indicating a column name (say, server latency) from a list of options populated from the RDBMS metadata. The user must not only choose just which fields and ranges from the database are rendered, but also choose image-specific parameters such as visualization type, axis labels, color codings, and so on.

Once the user has fully specified a visualization, the tool requests the necessary data by generating the appropriate SQL query and submitting it to the remote RDBMS. The RDBMS then returns the (relational) query results back to the visualization tool. Finally, the tool uses the fetched records to render the final visualization bitmap, which is displayed to the user. During these last three steps, the user waits idly for the visualization tool and the RDBMS.

When large datasets are being visualized, extremely long waits can negatively affect the analyst's level of engagement and ability to interactively produce successive visualizations [2], [6], [11]–[13]. As reported in Figure 2, our own experiments show that the industry-standard Tableau tool can take more than four minutes to produce a scatterplot on just 50M tuples fetched from an in-memory database.

Note that our sampling approach is not limited to the software architecture in Figure 3, as reducing the number of visualized records almost always brings performance benefits. Thus, even if engineers decide to combine the visualization and data management layers in the future, sampling-based methods will still be useful.

B. Data Sampling

Approximate query processing via sampling is a popular technique [17], [19]–[24], [28] for reducing the number of returned records, and random sampling or stratified sampling are two well-known methods for this. When using these methods, the visualization tool's query is run over a *sampled table(s)* (or simply, a *sample*) that is smaller than, and derived from, the original table(s). The sample(s) can be maintained by the same RDBMS. Since sampling approaches incur an additional overhead to produce the sample(s), these are typically performed in an *offline* manner [16], [21]: Once the sample is created and stored in the database, they can be interactively queried and visualized many times. (A sample can also be periodically updated when new data arrives [28].)

There is, of course, a tradeoff between output result quality and the size of the sample[1] (and thus, runtime). In the limit, a random sample of 100% of the original database will produce results with perfect fidelity, but will also not yield any reduction in runtime. Conversely, a sample of 0% of the database will yield a result with no fidelity, albeit very quickly. The exact size of the sample budget will be determined by deployment-specific details: the nature of the application, the patience of the user base, the amount of hardware resources available, and so on. As a result, the usefulness of a sampling method must be evaluated over a range of sample budgets, with a realistic measure of final output quality. Choosing a correct sampling budget is a known issue in the approximate query processing literature [16]. Of course, in any specific real-world deployment, we expect that the application will have a fixed maximum runtime or the size of a sample that the system must observe.

C. Visualization Quality

In this work, we focus on the production of *scatterplots* (including map plots, such as Figure 1) as one of the most popular visualization techniques. We leave other visualization types (such as bar charts, line charts, chloropleths, and so on) to future work.

Since the final value of a visualization is how much it helps the user, evaluating any sampling method means examining how users actually employ the visualizations they produce. Schneiderman, et al. proposed a taxonomy for information visualization types [29] and compiled some of common visualization-driven goals/tasks. Their list of goals included (i) regression, (ii) density estimation, and (iii) clustering. Our system aims to yield visualizations that help with each of these popular goals. We make no claim about other user goals and tasks for visualization, such as pattern finding and outlier detection, which we reserve for future work (although we have anecdotal evidence to suggest our system can address some of these tasks, too).

In this context, regression is the task of (visually) estimating the value of dependent variables given the value of independent variables. For example, if we want to know the temperature of the location specified by a pair of latitude and longitude coordinates, it belongs to the regression task. Density estimation is the task of understanding the distribution of the original data. For instance, one can use a map plot to understand the geometric area with the most cell phone subscribers. Clustering is a task that assigns data elements into distinct sets such that the data in the same group tend to be close to one another, while data in different groups are comparatively far apart.

Schneiderman, et al.'s list also included goals/tasks that are either poor fits for scatter plots, or are simply outside the scope of what we aim to accomplish in this paper: shape visualization (DNA or 3D structures), classification, hierarchy understanding, and community detection in networks. We explicitly do *not* attempt to produce visualizations that can help users with these tasks.

D. Our Approach

Our proposed method proceeds in two steps: (1) during *offline preprocessing*, we produce a sample that enable fast queries later, and (2) at *query time*, we choose a sample whose size is appropriate for the specific query.

Similar to any offline indexing technique, VAS also requires (1) the user to make choices about indexed columns and (2) an upfront computational work to speed up future queries. In other words, VAS can be considered as a specialized index designed for visualization workloads (e.g., Tableau). Note that, currently, even if users want to use offline indexing, there is no indexing technique that ensures fast and accurate visualizations, a problem solved by VAS.

Indexed columns can be chosen in three ways:

1) manually, by the DBA;
2) based on the most frequently visualized columns [16], [30]; or
3) based on statistical properties of the data [31].

Among these approaches, the second one is the simplest, works reasonably well in practice, and can be made resilient against workload changes [32]. Furthermore, note that visualization workloads, especially those supported by BI tools and SQL engines, are similar to exploratory SQL analytics (i.e., grouping, filtering, aggregations). Real-world traces from Facebook and Conviva [16] reveal that 80-90% of exploratory queries use 5-10% of the column combinations. Moreover, VAS only requires frequently visualized column pairs, not groups or filters.

The core innovation of our work is that we generate a sample according to a *visualization-specific metric*. That is, we believe that when a sample is generated according to the metric we propose in Section III below, the user will be able to accomplish their goals from Section II-C above (i.e., regression, density estimation, clustering) using the resulting visualization, even with a small number of rendered data points. We do *not*

---

[1]Here, the size of a sample means the number of the data points contained in the sample.

claim that our method will work for other visualization types, or even for other visualization goals. Indeed, our method of generating a sample could in principle be harmful to some goals (such as community detection tasks that require all members of a latent set to be sampled). However, scatter plots, map plots, and the three visualization goals we focus on are quite widespread and useful. Furthermore, modifying a visualization tool to only use our sampling method if a user declares an interest in one of these goals would be a straightforward task.

III. PROBLEM FORMULATION

The first step in our approach to obtain a good sample for scatter plot visualizations is defining a mathematical loss function that is closely correlated with the loss of visualization quality/utility from the user's perspective. Once we define a function that captures the visualization utility, our goal will be to solve an optimization problem; that is, finding a sample of a given size that has the minimum value for the loss function. In the rest of this section, we formally define our loss function and optimization problem. The algorithm for solving the derived optimization problem will be presented in Section IV. Also, in Section VI-B, we will report a comprehensive user study confirming that minimizing our loss function does indeed yield more useful visualizations for various user tasks.

We start our problem formulation by defining notations. We denote a dataset D of N tuples by $D = \{t_1, t_2, \ldots, t_N\}$. Each tuple t_i encodes the coordinate at which the associated *point* is displayed. For example, t_i is a pair of longitude and latitude in a map plot. A sample S is a subset of the dataset D and is denoted by $S = \{s_1, s_2, \ldots, s_K\}$. Naturally, s_i is one of t_j where $j = 1, \ldots, N$. The size of the sample S (which is denoted by K) is pre-determined based on the interactive latency requirement (see Section II-B) and is given as an input to our problem.

In designing our loss function, the objective is to measure the visualization quality degradation originating from the sampling process. The traditional goal of sampling in database systems is to maximize the number of tuples that match a selection predicate, particularly those on categorical attributes [16]. In contrast, the selection predicates of a scatter/map plot are on a continuous range, for which traditional approaches (e.g., uniform or stratified sampling) may not lead to high quality visualizations.

Therefore, to develop a more visualization-focused sampling technique, we first imagine a 2D space on which a scatter plot is displayed, and let x denote any of the points on the space. To measure the visualization quality loss, we make the following observations:

1) The visualization quality loss occurs, as the sample S does not include all tuples of D.
2) The quality loss at x is reduced as the sample includes points at or near x — two plots drawn using the original dataset (D) and the sample (S) might not look identical if S does not include a point at x whereas D includes one at x, but they will look similar if the sample contains points near x.

3) When there are already many points at or near x, choosing more points in that neighborhood does not significantly enhance a visualization.

To express the above observations in a formal way, we consider the following measure for visualization quality degradation at the point x:

$$\text{point-loss}(x) = \frac{1}{\sum_{s_i \in S} \kappa(x, s_i)}.$$

where $\kappa(x, s_i)$ is the function that captures the *proximity* between the two points, x and s_i. In this paper, we use $\kappa(x, s_i) = \exp(-\|x - s_i\|^2 / \epsilon^2)$ (see footnote[2] for ϵ) although other functions can also be used for κ if the function is a decreasing convex function of $\|x - s_i\|$ — the convexity is needed due to the third observation we described above. The equivalent quality metric can also be obtained by considering the problem as the regression problem that aims to approximate the original tuples in D using the sample S. See our technical report for an alternative derivation [34]. Note that the above loss value is reduced if there exists more sampled points near x where the proximity to x is captured by $\kappa(x, s_i)$. In other words, the visualization quality loss at x is minimized if S includes as many points as possible at and around x.

Note that the above loss function is defined for a single point x on the space on which a scatter/map plot is visualized, whereas the space on which a scatter plot is drawn has many of those points. As a result, our goal should be to obtain a sample S that minimizes the *combined* loss of all possible points on the space. Due to the reason, our objective is to find a sample S that minimizes the following expression:

$$\text{Loss}(S) = \int \text{point-loss}(x) \, dx = \int \frac{1}{\sum_{s_i \in S} \kappa(x, s_i)} \, dx \quad (1)$$

Here, the integration is performed over the entire 2D space.

Now, we perform several mathematical tricks to obtain an effectively equivalent but a more tractable problem, because the exact computation of the above integration requires an computationally expensive method such as a Monte Carlo experiment with a large number of points. Using a second-order Taylor expansion, we can obtain a simpler form that enables an efficient algorithm in the next section:

$$\min \int \frac{1}{\sum_{s_i \in S} \kappa(x, s_i)} \, dx$$
$$= \min \int 1 - (\sum \kappa(x, s_i) - 1) + (\sum \kappa(x, s_i) - 1)^2 \, dx$$
$$= \min \int (\sum \kappa(x, s_i))^2 - 3 \sum \kappa(x, s_i) \, dx$$
$$= \min \int \sum_{s_i, s_j \in S} \kappa(x, s_i)\kappa(x, s_j) \, dx$$

To obtain the last expression, we used the fact that the term $\int \sum \kappa(x, s_i) \, dx$ is constant since $\kappa(x, s_i)$ is a similarity function and we are integrating over every possible x, i.e., $\int \sum \kappa(x, s_i) \, dx$ has the same value regardless of the value of s_i. For the same reason, $\int \sum [\kappa(x, s_i)]^2 \, dx$ is also constant. By

---

[2]In our experiments, we set $\epsilon \approx \max(\|x_i - x_j\|)/100$ but there is a theory on how to choose the optimal value for ϵ as the only unknown parameter [33].

changing the order of integration and summation, we obtain the following optimization formulation, which we refer to as Visualization-Aware Sampling (VAS) problem in this paper.

Definition 1 (VAS). Given a fixed K, VAS is the problem of obtaining a sample S of size K as a solution to the following optimization problem:

$$\min_{S \subseteq D;\ |S|=K} \sum_{s_i, s_j \in S;\ i<j} \tilde{\kappa}(s_i, s_j)$$

$$\text{where } \tilde{\kappa}(s_i, s_j) = \int \kappa(x, s_i) \kappa(x, s_j) dx$$

In the definition above, we call the summation term $\sum \tilde{\kappa}(s_i, s_j)$ the *optimization objective*. With our choice of the proximity function, $\kappa(s_i, s_j) = \exp(-\|s_i - s_j\|^2/\epsilon^2)$, we can obtain a concrete expression for $\tilde{\kappa}(s_i, s_j)$: $\exp(-\|s_i - s_j\|^2/2\epsilon^2)$, after eliminating constant terms that do not affect the minimization problem. In other words, $\tilde{\kappa}(s_i, s_j)$ is in the same form as the original proximity function. In general, $\tilde{\kappa}(s_i, s_j)$ is another proximity function between the two points s_i and s_j since the integration for $\tilde{\kappa}(s_i, s_j)$ tends to have a larger value when the two points are close. Thus, in practice, it is sufficient to use any proximity function directly in place of $\tilde{\kappa}(s_i, s_j)$.

In the next section, we show that the VAS problem defined above is NP-hard and we present an efficient approximation algorithm for solving this problem. Later in Section VI-B we show that by finding a sample S that minimizes our loss function, we obtain a sample that, when visualized, best allows users to perform various visualization tasks.

IV. Solving VAS

In this section, we focus on solving the optimization problem derived in the previous section to obtain an optimal sample S. In the following section, we also describe how to extend the sample obtained by solving VAS to provide a richer set of information.

A. Hardness of VAS

First, we analyze the hardness of VAS formally.

Theorem 1. VAS (Problem 1) is NP-hard.

Proof: We show the NP-hardness of Problem 1 by reducing *maximum edge subgraph* problem to VAS.

Lemma 1. (*Maximum Edge Subgraph*) Given a undirected weighted graph $G = (V, E)$, choose a subgraph $G' = (V', E')$ with $|V'| = K$ that maximizes

$$\sum_{(u,v) \subset E'} w(u, v)$$

This problem is called *maximum edge subgraph*, and is NP-hard [35].

To reduce the above problem to VAS, the following procedure is performed: map i-th vertex v_i to i-th instance x_i, and set the value of $\tilde{\kappa}(x_i, x_j)$ to $w_{max} - w(v_i, v_j)$, where $w_{max} = \max_{v_i, v_j \subset V'} w(v_i, v_j)$. The reduction process takes

$O(|E| + |V|)$. Once the set of data points that minimize $\sum_{s_i, s_j \in X} \tilde{\kappa}(s_i, s_j)$ is obtained by solving VAS, we choose a set of corresponding vertices, and return them as an answer to the *maximum edge subgraph* problem. Since the *maximum edge subgraph* problem is NP-hard, and the reduction process takes a polynomial time, VAS is also NP-hard. ∎

Due to the NP-hardness of VAS, obtaining an exact solution to VAS is prohibitively slow, as we will empirically show in Section VI-D. Thus, in the rest of this section, we present an approximation algorithm for VAS (Section IV-B), followed by additional ideas for improvement (Section IV-B).

B. The Interchange Algorithm

In this section, we present our approximation algorithm, called Interchange. The Interchange algorithm starts from a randomly chosen set of size K and performs a replacement operation with a new data point if the operation decreases the optimization objective (i.e., the loss function). We call such a replacement, i.e., one that decreases the optimization objective, a *valid replacement*. In other words, Interchange tests for valid replacements as it sequentially reads through the data points from the dataset D.

One way to understand this algorithm theoretically is by imagining a Markov network in which each state represents a different subset of D where the size of the subset is K. The network then has a total of $\binom{D}{K}$ states. The transition between the states is defined as an exchange of one of the elements in the current subset S with another element in $D - S$. It is easy to see that the transition defined in this way is *irreducible*, i.e., any state can reach any other states following the transitions defined in such a way. Because Interchange is a process that continuously seeks a state with a lower optimization objective than the current one, Interchange is a hill climbing algorithm in the network.

Expand/Shrink procedure — Now we state how we can efficiently perform valid replacements. One approach to finding valid replacements is by substituting one of the elements in S with a new data point whenever one is read in, then computing the optimization objective of the set. For this computation, we need to call the proximity function $O(K^2)$ times as there are K elements in the set, and we need to compute a proximity function for every pair of elements in the set. This computation should be done for every element in S. Thus, to test for valid replacements, we need $O(K^3)$ computations for every new data point.

A more efficient approach is to consider only the part of the optimization objective for which the participating elements are *responsible*. We formally define the notion of responsibility as follows.

Definition 2. (*Responsibility*) The responsibility of an element s_i in set S is defined as:

$$\text{rsp}_S(s_i) = \frac{1}{2} \sum_{s_j \in S,\ j \neq i} \tilde{\kappa}(s_i, s_j).$$

Using the responsibility, we can speed up the tests for valid replacements in the following way. Whenever considering a new data point t, take an existing element s_i in S, and

Algorithm 1: Interchange algorithm.

```
input  : D = {t_1, t_2, ..., t_N}
output : A sample S of size K

   // set for pairs of (item, responsibility)
1  R ← ∅
2  foreach t_i ∈ D do
3  |   if |R| < K then  R ← Expand (R, t_i)
4  |   else
5  |   |   R ← Expand (R, t_i)
6  |   |   R ← Shrink (R)
7  |   end
8  end
9  S ← pick the first item of every pair in R
10 return S

11 subroutine Expand (R, t)
12 |   rsp ← 0      // responsibility
13 |   foreach (s_i, r_i) ∈ R do
14 |   |   l ← κ̃(t, s_i)
15 |   |   r_i ← r_i + l
16 |   |   rsp ← rsp + l
17 |   end
18 |   insert (t, rsp) into R
19 |   return R
20 end

21 subroutine Shrink (R)
22 |   remove (t, r) with largest r from R
23 |   foreach (s_i, r_i) ∈ R do
24 |   |   r_i ← r_i − κ̃(t, s_i)
25 |   end
26 |   return R
27 end
```

compute the responsibility of t in the set $S - \{s_i\} + \{t\}$. This computation takes $O(K)$ times. It is easy to see that if the responsibility of t in $S - \{s_i\} + \{t\}$ is smaller than the responsibility of s_i in the original set S, the replacement operation of s_i with the new data point t is a *valid replacement*. In this way, we can compare the responsibilities without computing all pairwise proximity functions. Since this test should be performed for every element in S, it takes a total of $O(K^2)$ computations for every new data point.

However, it is possible to make this operation even faster. Instead of testing for valid replacements by substituting the new data point t for one of the elements in S, we simply *expand* the set S by inserting t into the set, temporarily creating a set of size $K + 1$. In this process, the responsibility of every element in S is updated accordingly. Next, we find the element with the largest responsibility in the expanded set and remove that element from the set, shrinking the set size back to K. Again, the responsibility of every element in S should be updated. Algorithm 1 shows the pseudo-code for this approach. The theorem below proves the correctness of the approach.

Theorem 2. For $s_i \in S$, if replacing s_i with a new element t reduces the optimization objective of S, applying Expand followed by Shrink in Algorithm 1 replaces s_i with t. Otherwise, S remains the same.

Proof: Let $\tilde{\kappa}(S)$ indicate $\sum_{s_i, s_j \in S, i<j} \tilde{\kappa}(s_i, s_j)$. Also, define $S_- = S - \{s_i\}$ and $S_+ = S + \{t\}$. We show that

if the optimization objective before the replacement, namely $\tilde{\kappa}(S_- + \{s_i\})$, is larger than the optimization objective after the replacement, namely $\tilde{\kappa}(S_- + \{t\})$, then the responsibility of the existing element s_i in an expanded set, $\mathrm{rsp}_{S_+}(s_i)$, is also larger than the responsibility of the new element t in the expanded set, $\mathrm{rsp}_{S_+}(t)$. The proof is as follows:

$$\tilde{\kappa}(S_- + \{s_i\}) > \tilde{\kappa}(S_- + \{t\})$$
$$\iff \sum_{s_j \in S_-} \tilde{\kappa}(s_i, s_j) > \sum_{s_j \in S_-} \tilde{\kappa}(t, s_j)$$
$$\iff \tilde{\kappa}(s_i, t) + \sum_{s_j \in S_-} \tilde{\kappa}(s_i, s_j) > \tilde{\kappa}(s_i, t) + \sum_{s_j \in S_-} \tilde{\kappa}(t, s_j)$$
$$\iff \mathrm{rsp}_{S_+}(s_i) > \mathrm{rsp}_{S_+}(t).$$

Since the responsibility of s_i is larger than that of t in the expanded set S_+, the Shrink routine will remove s_i. If no element exists whose responsibility is larger than that of t, then t is removed by this routine and S remains the same. ∎

In both the Expand and Shrink routines, the responsibility of each element is updated using a single loop, so both routines take $O(K)$ computations whenever a new data point is considered. Thus, scanning the entire dataset and applying these two routines will take $O(NK)$ running time.

The Interchange algorithm, if it runs until no replacement decreases the optimization objective, has the following theoretical bound.

Theorem 3. Let's say that the sample obtained by Interchange is S_{int}, and the optimal sample is S_{opt}. The quality of S_{int}, or the optimization objective, has the following upper bound:

$$\frac{1}{K(K-1)} \sum_{s_i, s_j \in S_{int};\ i<j} \tilde{\kappa}(s_i, s_j)$$
$$\leq \frac{1}{4} + \frac{1}{K(K-1)} \sum_{s_i, s_j \in S_{opt};\ i<j} \tilde{\kappa}(s_i, s_j)$$

In the expression above, we compare the difference between the averaged optimization objectives.

Proof: Due to the submodularity of VAS, which we show in our technical report [34], we can apply the result of Nemhauser, et al. [36] and obtain the result above. ∎

Ideally, Interchange should be run until no more valid replacements are possible. However, in practice, we observed that even running the algorithm for half an hour produces a high quality sample. When more time is permitted, the algorithm will continuously improve the sample quality until convergence.

Speed-Up using the Locality of Proximity function — Proximity functions such as $\exp(-\|x - y\|^2/\epsilon^2)$ have a property called *locality*. The locality property of a proximity function indicates that its value becomes negligible when the distance between the two data points is not close—an idea also used in accelerating other algorithms [37]. For example, our proximity function value is 1.12×10^{-7} when the distance between the two points is 4ϵ; thus, even though we ignore pairs whose distance is larger than a certain threshold, it will not affect the final outcome much. Exploiting this property, we can make the Expand and Shrink operations much faster by only

considering the data points that are close enough to new data points. For a proximity check, our implementation used R-tree.

V. EXTENDING VAS: EMBEDDING DENSITY

VAS aims to minimize a visualization-driven quality loss, yielding scatter/map plots that are highly similar to those generated by visualizing the entire dataset. However, we need a different strategy if the user's intention is to estimate the density or find clusters from the scatter plot. This is because humans cannot visually distinguish multiple data points on a scatter plot if they are duplicates or extremely close to one another. This can make it hard to visually estimate the number or density of such data points. One way to address this is to account for the number of near-duplicate points in each region. For example, points drawn from a dense area can be plotted with a larger legend size or some jitter noise can be used to provide additional density in the plot. In other words, the font size of each point or the amount of jitter noise will be proportional to the original density of the region the point is drawn from. VAS can be easily extended to support such an approach, as follows:

1) Obtain a sample using our algorithm for VAS.
2) Attach a counter to every sampled point.
3) While scanning the dataset once more, increase a counter if its associated sampled point is the nearest neighbor of the data point that was just scanned.

With these extra counters, we can now visualize the density of areas (each of which is represented by its nearest sampled point), e.g., using different dot sizes or by adding jitter noise in proportion to each point's density. (See Section VI-B for a scatter plot example.)

Note that the above process only adds an extra column to the database and, therefore, does not alter our basic Interchange algorithm for VAS. Also, this extension does not require any additional information from users.

Note that, for the above density embedding process, a special data structure such as a k-d tree [38] can be used to make the nearest neighbor tests more efficient. This is done by using the sample obtained in the first pass to build a k-d tree, then using the tree to identify the nearest data points in the sample during the second pass. Since k-d trees perform the nearest neighbor search in $O(\log K)$, the overall time complexity for the second pass is $O(N \log K)$.

VI. EXPERIMENTS

We run four types of experiments to demonstrate that VAS and VAS with density embedding can produce high-quality plots in less time than competing methods.

1) We study the runtime of existing visualization systems that were introduced in Figure 2.
2) In a user study, we show that users were more successful when they used visualizations produced by VAS than with other methods. We also show that user success and our loss function were negatively correlated (that is, users were successful when our loss function is minimized).
3) We show that VAS could obtain a sample of a fixed quality level (that is, loss function level) with fewer data
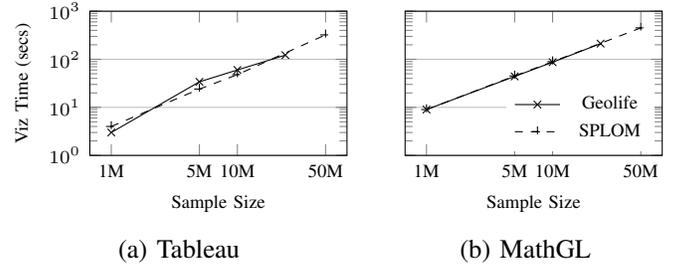

(a) Tableau   (b) MathGL

Fig. 4: Time to produce plots of various sizes using existing visualization systems.

points than competing methods. We demonstrate this over a range of different datasets and sample quality levels.

4) We empirically study the Interchange algorithm: we compare its quality and runtime to those of the exact method, examine the relationship between runtime and sample quality, and investigate the impact of our optimization on runtime.

All of our experiments were performed using two datasets: the Geolife dataset and the SPLOM dataset. The Geolife dataset was collected by Microsoft Research [26]. It contained latitude, longitude, elevation triples from GPS loggers, recorded mainly around Beijing. Our full database contained 24.4M tuples. We also used SPLOM, a synthetic dataset generated from several Gaussian distributions that had been used in previous visualization projects [4], [39]. We used parameters identical to previous work, and generated a dataset of five columns and 1B tuples. We performed our evaluations on an Amazon EC2 memory instance (r3.4xlarge) which contained 16 cores and 122G memory.

A. Existing Systems are Slow

Our discussions in this paper are based on the idea that plotting a scatter plot using an original dataset takes an unacceptably long period of time. We tested two state-of-the-art systems: Tableau [40] and MathGL [41]. Tableau is one of the most popular commercial visualization software available on Windows, and MathGL is an open source scientific plotting library implemented in C++. We tested both the Geolife and SPLOM datasets. The results are shown in Figure 4.

In both systems, the visualization time includes (1) the time to load data from SSD storage (for MathGL) or from memory (for Tableau) and (2) the time to render the data into a plot. We can see that even when the datasets contained just 1M records, the visualization time was more than the 2-second interactive limit. Moreover, visualization time grew linearly with sample size.

B. User Success and Sample Quality

In this section we make two important experimental claims about user interaction with visualizations produced by our system. First, users are more successful at our specified goals when using VAS-produced outputs than when using outputs from uniform random sampling or stratified sampling. Second, user success and our loss function — that is, our measure
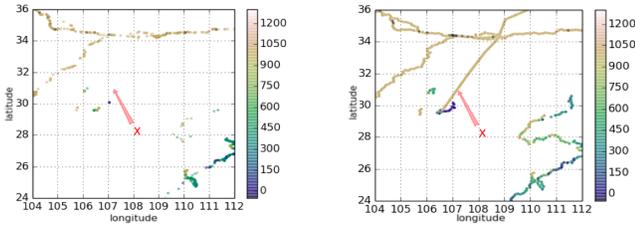
(a) Stratified Sampling   (b) VAS

Fig. 5: Example figures used in the user study for the regression task. We asked the altitude of the location pointed by 'X'. The left was generated by stratified sampling and the right was generated by VAS.
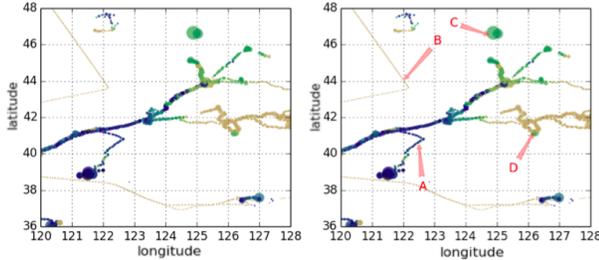

Fig. 6: An example figure used in the user study for the density estimation task. This figure was generated using VAS with density embedding. The left-hand image is the original figure. The right-hand image contains four test markers, used to ask users to choose the densest area and the sparsest areas.

Sample size	Uniform	Stratified	VAS
100	0.213	0.225	0.428
1,000	0.260	0.285	0.637
10,000	0.215	0.360	0.895
100,000	0.593	0.644	0.989
Average	0.319	0.378	**0.734**

(a) Regression

Sample size	Uniform	Stratified	VAS	VAS w/ density
100	0.092	0.524	0.323	0.369
1,000	0.628	0.681	0.311	0.859
10,000	0.668	0.715	0.499	0.859
100,000	0.734	0.627	0.455	0.869
Average	0.531	0.637	0.395	**0.735**

(b) Density Estimation

Sample size	Uniform	Stratified	VAS	VAS w/ density
100	0.623	0.486	0.521	0.727
1,000	0.842	0.412	0.658	0.899
10,000	0.931	0.543	0.845	0.950
100,000	0.897	0.793	0.864	0.965
Average	0.821	0.561	0.722	**0.887**

(c) Clustering

The uniform random sampling method chooses K data points purely at random, and as a result, tends to choose more data points from dense areas. We implemented the single-pass reservoir method for simple random sampling. Stratified sampling divides a domain into non-overlapping bins and performs uniform random sampling for each bin. Here, the number of the data points to draw for each bin is determined in the most balanced way. For example, suppose there are two bins and we want a sample of size 100. If there are enough data points to sample from those two bins, we sample 50 data points from each bin. Otherwise, if the second bin only has 10 available data points, then we sample 90 data points from the first bin, and 10 data points from the second bin. Stratified sampling is a straightforward method that avoids uniform random sampling's major shortcoming (that is, uniform random sampling draws most of its data points from the densest areas). In our experiment, stratified sampling divided the domain of Geolife into 100 exclusive bins and performed uniform random sampling for each bin using the reservoir method.

Table I(a) summarizes user success in the regression task. The result shows that users achieved the highest accuracy in the regression task when they used VAS, significantly outperforming other sampling methods.

Density Estimation — For the density estimation task, we created samples whose sizes ranged 100-100K using four different sampling methods: uniform random sampling, stratified sampling, VAS, and VAS with density embedding. Using those samples, we chose 5 different zoomed-in areas. For each zoomed-in area, we asked users to identify the densest and the sparsest areas among 4 different marked locations. Figure 6

of sample quality — are correlated. We validate these claims with a user study performed using Amazon's Mechanical Turk system.

1) User Success: We tested three user goals: regression, density estimation, and clustering.

Regression — To test user success in the regression task, we gave each user a sampled visualization from the Geolife data. We asked the users to estimate the altitude at a specified latitude and longitude. Naturally, the more sampled data points that are displayed near the location in question, the more accuracy users are likely to achieve. Figure 5 shows two examples of test visualizations given to users for the regression task (users were asked to estimate the altitude of the location marked by 'X'). We gave each user a list of four possible choices: the correct answer, two false answers, and "I'm not sure".

We tested VAS, random uniform sampling, and stratified sampling. We generated a test visualization for each sampling method at four distinct sample sizes ranging from 100 to 100K. For each test visualization, we zoomed into six randomly-chosen regions and picked a different test location for each region. Thus, we had 72 unique test questions (3 methods * 4 sample sizes * 6 locations). We gave each package of 72 questions to 40 different users and averaged the number of correct answers over each distinct question. To control for worker quality, we filtered out users who failed to correctly answer a few trivial "trapdoor" questions.

shows an example visualization shown to a test user. As a result, we generated 80 unique visualizations. We again posed the package of 80 questions to 40 unique users, and again filtered out users who failed to answer easy trapdoor questions.

The result of the density estimation task is shown in Table I(b). Interestingly, the basic VAS method *without* density estimation yielded very poor results. However, when we augmented the sample with density embedding, users obtained even better success than with uniform random sampling. One of the reasons that 'VAS with density' was superior to uniform random sampling was because we not only asked the users to estimate the densest area, but also asked them to estimate the sparsest area of those figures. The figures generated by uniform random sampling typically had few points in sparse areas, making it difficult to identify the sparsest area.

Clustering — Lastly, we compared user performance in the clustering task. Since the Geolife dataset did not have ground-truth for clustering, we used synthetic datasets that we generated using Gaussian distributions instead. Using two-dimensional Gaussian distributions with different covariances, we generated 4 datasets, 2 of which were generated from 2 Gaussian distributions and the other 2 were generated from a single Gaussian distribution. (This dataset was similar to SPLOM, which unfortunately has a single Gaussian cluster, making it unsuitable for this experiment.)

Using the same 4 sampling methods that were used in the density estimation task, we created samples whose sizes ranged 100-100K, and tested if users could correctly identify the number of underlying clusters given the figures generated from those samples. In total, we created 64 questions (4 methods, 4 datasets, and 4 sample sizes). We again asked 40 Mechanical Turk users (or simply Turkers) and filtered out bad workers.

Table I(c) summarizes the result of the clustering task. As in the density estimation task, 'VAS with density' allowed users to be more successful than they were with visualizations from uniform random sampling. Although VAS without density did not perform as well as uniform random sampling, it produced a roughly comparable score.

We think the reason VAS *without* density estimation showed comparable performance was that we used no more than 2 Gaussian distributions for data generation, and the Turkers could recognize the number of the underlying clusters from the outline of the sampled data points. For example, if the data were generated from two Gaussian distributions, the data points sampled by VAS would look like two partially overlapping circles. The Turkers would have shown poorer performance if there was a cluster surrounded by other clusters.

On the other hand, stratified sampling did poorly in this clustering task because it performed a separate random sampling for each bin, i.e., the data points within each bin tend to group together, and as a result, the Turkers found that there were more clusters than actually existed.

2) Correlation with Sample Quality: In this section, we test whether the VAS's optimization criterion of $\mathrm{Loss}(S)$ had a close relationship to our visualization users' success in reaching their end goals. If they were highly correlated, we have some empirical evidence that samples which minimize the VAS loss function will yield useful plots.
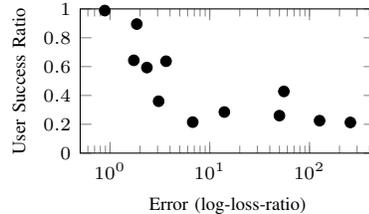

Fig. 7: The relationship between the loss and user performance on the regression task. The samples with smaller losses resulted in better success ratios in general in the regression task.

In particular, we examined this relationship for the case of regression. For each combination of sample size and sampling method, we produced a sample and corresponding visualization. We computed $\mathrm{Loss}(S)$ using the expression in Equation 1. We then measured the correlation between the *loss* and average user performance on the regression task for that visualization.

To compute the loss (Equation 1), which includes integration, we used the Monte Carlo technique using 1,000 randomly generated points in the domain of the Geolife dataset. For this process, we determined that randomly generated points were within the domain if there existed any data point in the original dataset whose distance to the randomly generated data points was no larger than 0.1. Now, the integral expression was replaced with a summation as follows:

$$\mathrm{Loss}(S) = \frac{1}{1000} \sum_{i=1}^{1000} \frac{1}{\sum_{s_i \in S} \kappa(x_i, s_i)}.$$

This *loss* computed above is the *mean* of one thousand values. One problem we encountered in computing the mean was that the point-loss often became so large that `double` precision could not hold those quantities. To address this, we used the *median* instead of the *mean* in this section because the median is less sensitive to outliers. Note that the median is still valid for a correlation analysis because we did not observe any case where a sample with a larger mean has a smaller median compared to another sample.

Next, to compare loss in a more consistent way, we computed the following quantity:

$$\mathrm{log\text{-}loss\text{-}ratio}(S) = \log_{10} \left[\frac{\mathrm{Loss}(S)}{\mathrm{Loss}(D)} \right]$$

where D is the original dataset. $\mathrm{Loss}(D)$ is the lowest *loss* that a sample can achieve; thus, samples with log-loss-ratios close to zero can be regarded as good samples based on this metric.

Next we examined the relationship between a sample's log-loss-ratio and the percentage of the questions that were correctly answered in the regression task using the sample's corresponding visualization. If the two metrics yield similar rankings of the sampled sets, then the VAS optimization criterion is a good guide for producing end-user visualizations. If the two metrics yield uncorrelated rankings, then our VAS criterion is faulty.

Figure 7 depicts the results. The figure clearly shows the negative correlation between the loss and user success ratio in
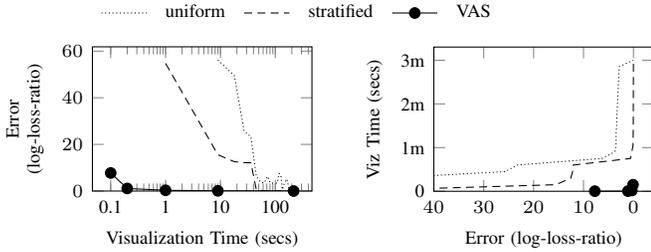
(a) Error given time   (b) Time given error

Fig. 8: Relationship between visualization production time and error for the three sampling methods.

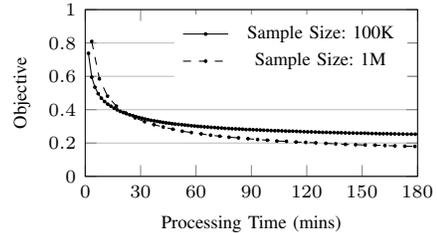| N	Metric	MIP	Approx. VAS	Random
50	Runtime	1m 7s	0s	0s
	Opt. objective	0.160	0.179	3.72
	$Loss(S)$	1.5e+26	1.5e+26	2.5e+29
60	Runtime	1m 33s	0s	0s
	Opt. objective	0.036	0.076	3.31
	$Loss(S)$	3.8e+11	1.6e+16	2.5e+29
70	Runtime	14m 26s	0s	0s
	Opt. objective	0.047	0.048	3.02
	$Loss(S)$	1.8e+13	1.8e+13	9.45e+33
80	Runtime	48m 55s	0s	0s
	Opt. objective	0.043	0.048	2.25
	$Loss(S)$	8.5e+13	1.8e+13	9.4e+35

Fig. 9: Processing Time vs. Quality. The lower the objective, the higher the quality is. The Interchange algorithm for VAS produces a high-quality visualization in a relatively short period of time. The quality is improved incrementally as more processing time is allowed.

the regression task. Because the X-axis of the figure is the loss function that we aim to minimize to obtain a good sample, the negative correlation between the two metrics shows the validity of our problem formulation.

Also, when we computed Spearman's rank correlation coefficient[3], the correlation coefficient was -0.85, indicating a strong negative correlation between user success and the log-loss-ratio. (Its p-value was 5.2×10^{-4}.) Put another way, minimizing our loss function for a sample should do a good job of maximizing user success on the resulting visualizations. This result indicates that the problem formulation in Section III and the intuition behind it was largely valid.

C. VAS Uses a Smaller Sample

This section shows that VAS can produce a better sample than random uniform sampling or stratified sampling. That is, for a fixed amount of visualization production time, its quality (loss function value) is lower; or, that for a fixed quality level (loss function value), it needs less time to produce the visualization. (The visualization production time is linear with the number of data points.)

We used the Geolife dataset and produced samples of various sizes (and thus, different visualization production times). Figure 8(a) shows the results when we varied the visualization time: VAS always produced a sample with lower loss function values (i.e., higher quality) than other methods. The quality gap between the methods did not become smaller until after an entire minute of runtime. We show a similar result with the other dataset in our technical report [34]

Figure 8(b) shows the same data using a different perspective. We fixed the loss function value (quality) and measured how long it takes to visualize the corresponding samples. Because the samples generated by our method had much smaller losses compared to other methods, all of the data points in the figure are in the bottom right corner. Competing methods required much more time than VAS to obtain the same quality (loss function value).

---

[3]Spearman's rank correlation coefficient produces -1.0 for pairs of variables that are completely negatively correlated, and 1.0 for pairs of variables that are completely positively correlated.

D. Algorithmic Details

We now examine three internal qualities of the VAS technique: approximate vs. exact solution, runtime analysis, and optimization contributions.

Exact vs. Approximate — The NP-hardness of VAS supports the need for an approximation algorithm. This section empirically examines the NP-hardness of VAS.

We think one of the best options for obtaining an exact solution to VAS is by converting the problem to an instance of integer programming and solving it using a standard library. Refer to our report [34] for converting VAS to an instance of Mixed Integer Programming (MIP). We used the GNU Linear Programming Kit [42] to solve the converted MIP problem.

Table II shows the time it took to obtain exact solutions to VAS with datasets of very small sizes. The sample size K was fixed to 10 in all of the experiments in the table. According to the result, the exact solutions to VAS showed better quality, but the procedure to obtain them took considerably longer. As shown, obtaining an exact solution when $N = 80$ took more than 40 minutes, whereas the time it took by other sampling methods was negligible. Clearly, the exact solution is not feasible except for extremely small data sizes.

Runtime Analysis — VAS gradually improves its sample quality as more data is read and processed. As observed in many statistical optimization routines, VAS offers good-quality plots long before reaching its optimal state. To investigate this phenomenon, we measured the relationship between

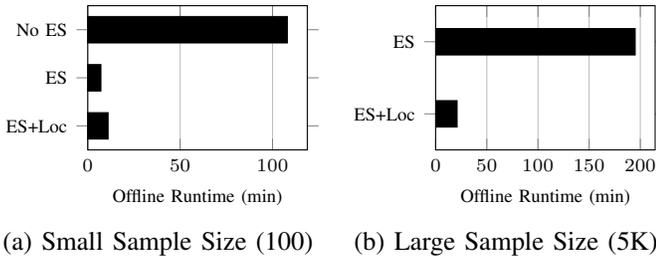(a) Small Sample Size (100)   (b) Large Sample Size (5K)

Fig. 10: Runtime comparison of different levels of optimizations. For this experiment, we used the Geolife dataset. ES+Loc indicates that both Expand/Shrink (ES) operation and the locality of a proximity function were used.

"processing time" and "visualization quality." The result is shown in Figure 9. Note that the Y-axis of the figure is the objective[4] of our minimization problem; thus, the lower the objective, the higher the associated visualization's quality. In this experiment, we used the Geolife dataset. Figure 9 demonstrates that our Interchange algorithm improved the visualization quality quickly at its initial stages, and the improvement rate slowed down gradually. Notably, VAS produced low-error plots within only tens of minutes of processing time. The storage overhead of our algorithm is only $O(K)$, where K is the sample size.

Optimization Contribution — To quantify the impact of our optimization efforts on the runtime reduction, we measured the runtime of three different settings:

1) No Expand/Shrink (No ES): This is the most basic configuration that does not use the Expand/Shrink approach, but instead compares the responsibility when a new point is switched with another one in the sample.
2) Expand/Shrink (ES): This version uses the Expand/Shrink operation, reducing the time complexity by $O(K)$, where K is the sample size.
3) Expand/Shrink+Locality (ES+Loc): This version uses an additional R-tree to speed up the Expand/Shrink operations. This version is possible due to the locality of our loss function.

Figure 10 shows the results. When the sample size was relatively small (100), the second approach (Expand/Shrink), which does not exploit the locality, showed the shortest runtime due to no extra overhead coming from maintaining an extra R-tree data structure. However, when the sample size was relatively large (5K), the last approach (ES+Loc) that exploits the locality of the loss function showed the fastest runtime. When the user is interested in large samples (more than 10K at least), the last approach that uses R-tree to exploit locality will be the most preferable choice. The runtime sensitivity to sample size suggests that in the future, it may be useful to employ an optimizer that chooses the most appropriate algorithm setting, given a requested sample size.

---

[4]We scaled the objectives appropriately for a clearer comparison.

VII. RELATED WORK

Support for interactive visualization of large datasets is a fast-growing area of research interest [1]–[10], [15], along with other approximate techniques for interactive processing of non-conventional queries [43]. Most of the work to date has originated from the user interaction community, but researchers in database management have begun to study the problem. Known approaches fall into a few different categories.

The most directly related work is that of Battle, et al. [15]. They proposed ScalaR, a system for *dynamic reduction* of query results that are too large to be effectively rendered on-screen. The system examines queries sent from the visualization system to the RDBMS and if necessary, inserts aggregation, sampling, and filtering query operators. ScalaR uses simple random sampling, and so could likely be improved by adopting our sampling technique. For bar graphs, Kim et al. [44] proposed an order-preserving sampling method, which examines fewer tuples than simple random sampling.

Binned aggregation approaches [3]–[5] reduce data by dividing a data domain into tiles or bins, which correspond to materialized views. At visualization time, these bins can be selected and aggregated to produce the desired visualization. Unfortunately, the exact bins are chosen ahead of time, and certain operations — such as zooming — entail either choosing a very small bin size (and thus worse performance) or living with low-resolution results. Because binned aggregation needs to pre-aggregate all the quantities in advance, the approach is less flexible when the data is changing, such as measured temperatures over time; our method does not have such a problem.

Wickham [3] proposed to improve visualization times with a mixture of binning and summarizing (very similar to binned aggregation) followed by a statistical smoothing step. The smoothing step allows the system to avoid problems of high variability, which arise when the bins are small or when they contain eccentric values. However, the resulting smoothed data may make the results unsuitable for certain applications, such as an outlier finding. This smoothing step itself is orthogonal to our work, i.e., when there appears to be high variability in the sample created by our proposed method, the same smoothing technique can be applied to present more interpretable results. The smoothing process also benefits from our method because VAS creates a sample much smaller than the original database, thus, makes smoothing faster. The *abstract rendering pipeline* [1] also maps bins to regions of data, but the primary goal of this system is to modify the visualization, not performance.

Parallel rendering exploits parallelism in hardware to speed up visual drawing of the visualization [9], [10]. It is helpful but largely orthogonal to our contributions. SEEDB is a system that discovers the most interesting bar graphs [45] from datasets.

Incremental visualization proposes a streaming data processing model, which quickly yields an initial low-resolution version of the user's desired bitmap [7], [8]. The system continues to process data after showing the initial image and progressively refines the visualization. When viewed in terms of our framework in Section II, this method amounts to increasing the sample budget over time and using the new

samples to improve the user's current visualization. Thus, incremental visualization and sample-driven methods should benefit from each other.

VIII. CONCLUSIONS AND FUTURE WORK

We have described the VAS method for visualization data reduction. VAS is able to choose a subset of the original database that is very small (and thus, fast) while still yielding a high-quality scatter or map plot. Our user study showed that for three common user goals — regression, density estimation, and clustering — VAS outputs are substantially more useful than other sampling methods' outputs with the same number of tuples.

We believe our core topic — data system support for visualization tools — is still in its infancy and entails a range of interesting challenges. In particular, we plan to investigate techniques for rapidly generating visualizations for other user goals (including outlier detection, trend identification) and other data types (such as large networks).

ACKNOWLEDGMENT

REFERENCES

[1] J. A. Cottam, A. Lumsdaine, and P. Wang, "Overplotting: Unified solutions under abstract rendering," in *BigData Conference*, 2013.

[2] J. Heer and S. Kandel, "Interactive analysis of big data," *XRDS*, 2012.

[3] H. Wickham, "Bin-summarise-smooth: a framework for visualising large data," had.co.nz, Tech. Rep., 2013.

[4] Z. Liu, B. Jiang, and J. Heer, "immens: Real-time visual querying of big data," in *Computer Graphics Forum*, vol. 32, 2013.

[5] L. D. Lins, J. T. Klosowski, and C. E. Scheidegger, "Nanocubes for real-time exploration of spatiotemporal datasets," *TVCG*, 2013.

[6] M. Barnett, B. Chandramouli, R. DeLine, S. M. Drucker, D. Fisher, J. Goldstein, P. Morrison, and J. C. Platt, "Stat!: an interactive analytics environment for big data," in *SIGMOD*, 2013.

[7] 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*, 2012.

[8] D. Fisher, S. M. Drucker, and A. C. König, "Exploratory visualization involving incremental, approximate database queries and uncertainty," *IEEE Computer Graphics and Applications*, vol. 32, no. 4, pp. 55–62, 2012.

[9] J. A. Cottam and A. Lumsdaine, "Automatic application of the data-state model in data-flow contexts," in *IV*, 2010, pp. 5–10.

[10] H. Piringer, C. Tominski, P. Muigg, and W. Berger, "A multi-threading architecture to support interactive visual exploration," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 6, pp. 1113–1120, 2009.

[11] R. B. Miller, "Response time in man-computer conversational transactions," in *fall joint computer conference*, 1968.

[12] B. Shneiderman, "Response time and display rate in human performance with computers," *CSUR*, 1984.

[13] Z. Liu and J. Heer, "The effects of interactive latency on exploratory visual analysis," 2002.

[14] I. Stoica, "For big data, moore's law means better decisions," http://www.tableausoftware.com/.

[15] L. Battle, M. Stonebraker, and R. Chang, "Dynamic reduction of query result sets for interactive visualizaton," in *BigData Conference*, 2013, pp. 1–8.

[16] 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 *EuroSys*, 2013.

[17] B. Mozafari, "Verdict: A system for stochastic query planning," in *CIDR, Biennial Conference on Innovative Data Systems*, 2015.

[18] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo, "The analytical bootstrap: a new method for fast error estimation in approximate query processing," in *SIGMOD*, 2014.

[19] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, "The aqua approximate query answering system," in *SIGMOD*, 1999.

[20] B. Babcock, S. Chaudhuri, and G. Das, "Dynamic sample selection for approximate query processing," in *SIGMOD*, 2003.

[21] S. Chaudhuri, G. Das, and V. Narasayya, "Optimized stratified sampling for approximate query processing," *TODS*, 2007.

[22] J. M. Hellerstein, P. J. Haas, and H. J. Wang, "Online aggregation," *SIGMOD*, 1997.

[23] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra, "Scalable approximate query processing with the dbo engine," *TODS*, 2008.

[24] C. Olston, E. Bortnikov, K. Elmeleegy, F. Junqueira, and B. Reed, "Interactive analysis of web-scale data." in *CIDR*, 2009.

[25] B. Mozafari and N. Niu, "A handbook for building an approximate query engine," *IEEE Data Engineering Bulletin*, 2015.

[26] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding mobility based on gps data," in *Ubiquitous computing*, 2008.

[27] "Tableau for the enterprise: An overview for it," http://www.tableausoftware.com/sites/default/files/whitepapers/whitepaper_tableau-for-the-enterprise_0.pdf.

[28] S. Agarwal, A. Panda, B. Mozafari, A. P. Iyer, S. Madden, and I. Stoica, "Blink and it's done: Interactive queries on very large data," *PVLDB*, 2012.

[29] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Visual Languages*, 1996.

[30] S. Idreos, M. L. Kersten, and S. Manegold, "Database cracking." in *CIDR*, 2007.

[31] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina, "Seedb: Visualizing database queries efficiently," *PVLDB*, 2013.

[32] B. Mozafari, E. Z. Y. Goh, and D. Y. Yoon, "CliffGuard: A principled framework for finding robust database designs," in *SIGMOD*, 2015.

[33] P. Cizek, W. K. Härdle, and R. Weron, *Statistical tools for finance and insurance*. Springer, 2005.

[34] Y. Park, M. Cafarella, and B. Mozafari, "Our technical report," http://arxiv.org/abs/1510.03921.

[35] U. Feige, D. Peleg, and G. Kortsarz, "The dense k-subgraph problem," *Algorithmica*, vol. 29, 2001.

[36] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions-I," *Mathematical Programming*, vol. 14, 1978.

[37] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *JMLR*, vol. 9, 2008.

[38] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of ACM*, vol. 18, 1975.

[39] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer, "Profiler: Integrated statistical analysis and visualization for data quality assessment," in *AVI*, 2012.

[40] "Tableau software," http://www.tableausoftware.com/.

[41] "Mathgl," http://mathgl.sourceforge.net/doc_en/Main.html.

[42] A. Makhorin, "Glpk (gnu linear programming kit), version 4.54," *http://www.gnu.org/software/glpk*.

[43] Y. Park, M. Cafarella, and B. Mozafari, "Neighbor-sensitive hashing," *PVLDB*, 2015.

[44] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld, "Rapid sampling for visualizations with ordering guarantees," *PVLDB*, 2015.

[45] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis, "Seedb: Efficient data-driven visualization recommendations to support visual analytics," *PVLDB*, 2015.