Visualization-Aware Sampling for Very Large Databases

Yongjoo Park
University of Michigan
Ann Arbor
pyongjoo@umich.edu

Michael Cafarella
University of Michigan
Ann Arbor
michjc@umich.edu

Barzan Mozafari
University of Michigan
Ann Arbor
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 tuples which minimize a visualization-inspired loss function. Our user study confirms that optimizing this loss function correlates strongly with user success in using the resulting visualizations. We also show the NP-hardness of our optimization problem and propose an efficient approximation algorithm. Our experiments show that, compared to previous methods, (i) using the same sample size, 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× faster.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

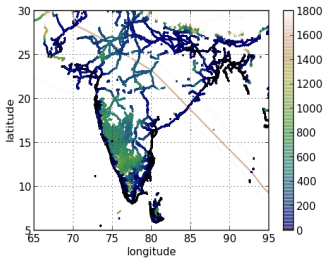
Visualization-Aware Sampling, approximate visualization, approximation algorithm, submodularity

1. INTRODUCTION

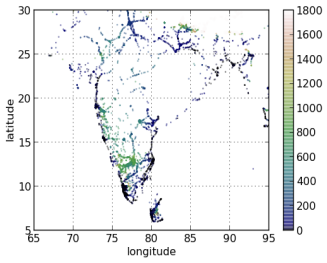
Data scientists frequently rely on visualizations for analyzing data and gleaning insight. For productive data exploration, analysts should be able to produce *ad hoc* visualizations in interactive time (a well-established goal in the visualization and human-computer interaction (HCI) community [5,11,12,14–16,21,23,30,37]). 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 6.1 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 [22,26,31]. Unfortunately, dataset sizes are already growing faster than Moore's Law [33] (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 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 1a 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 as color). Another example of a map plot would be if a climate scientist who wants to study temperatures measured at various sensor locations rendered each sensor reading as a colored dot on a 2D plane.

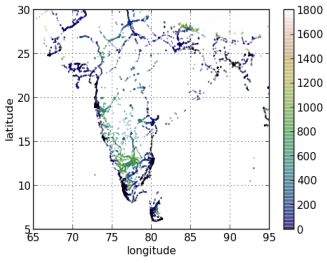
One approach for reducing the time involved in visualization production is via *data reduction* [6]. Reducing the dataset reduces the amount of work for the visualization system (by reducing the CPU, I/O, and rendering times) but at a possible cost to the quality of output visualization. Effective data reduction will shrink the dataset as much as possible while still producing an output that is faithful to the unreduced original. Sampling is a popular database method for reducing the amount of data to be processed, often in the context of approximate query processing [2–4, 9, 17, 18, 29].



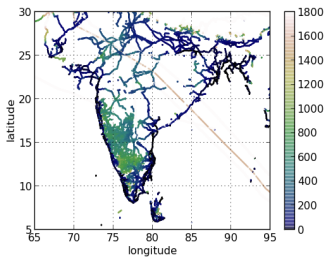
(a) Original 2B-tuple dataset, visualized in 71 mins.



(b) 1M-tuple uniform sample, visualized in 3 secs.



(c) 1M-tuple stratified sample, visualized in 3 secs.



(d) 1M-tuple VAS sample, visualized in 3 secs.

Figure 1: Different sampled map plots of a database of India GPS device trace data from the OpenStreetMap project. Each colored dot represents the elevation sensed at a specific latitude and longitude.

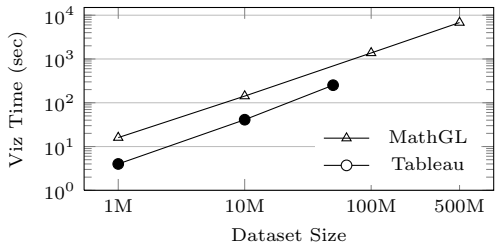


Figure 2: The latency for generating scatter plot visualizations using Tableau and MathGL (a library for scientific graphics).

While uniform (random) sampling and stratified sampling are two of the most common and effective approaches in approximate query processing, 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 [23]. For example, Figures 1b and 1c show the India region of the small uniform and stratified samples of OpenStreetMap dataset. The resulting visualizations are relatively unfaithful to the original, partially because they have chosen to sample many "visually similar" tuples that do not make a large impact on the output. If the user chooses to zoom in on a region of interest, the quality degrades even further. Analysts who use these fast but low-quality visualizations may draw incorrect conclusions.

Previous Approaches — Architecturally, our system is similar to ScalaR [6], 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

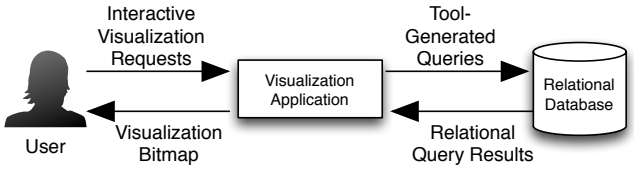


Figure 3: Standard model of user interaction with the combined visualization and database system.

for improving visualization time, including binned aggregation [21, 23, 37], parallel rendering [11, 30], and incremental visualization [14, 15]. 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 1d shows the plot generated by our proposed method, called Visualization-Aware Sampling (VAS). Using the same number of tuples as random and stratified sampling, VAS yields a much higher-fidelity result. 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-specific, variance-based metric that we have found correlates well with user success across a range of scatter and map plot tasks.

Contributions — This paper makes the following contributions:

- We define the notion of VAS as an optimization problem (Section 3).
- We prove that the VAS problem is NP-hard and an offer efficient approximation algorithm. We establish an worst-case guarantee for our approximate solution (Section 4).
- 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 with competing approaches, but using up to 400× fewer data points. Alternatively, if VAS can process an equal number of samples as competing methods, it can deliver a visualization with a significantly higher quality (Section 6).

Finally, we cover related work in Section 7 and conclude with a discussion of future work in Section 8.

2. SYSTEM OVERVIEW

2.1 Software Architecture Model

Figure 3 shows the software architecture model that we focus on in this paper. This is a standard architecture sup-

ported by the popular Tableau system [1]. It is also similar to ScalaR's "dynamic reduction" software architecture [6]. 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 choose not just which fields and ranges from the database are rendered, but also 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, the extremely long waits can negatively affect the analyst's level of engagement and ability to interactively produce successive visualizations [5, 16, 22, 26, 31]. As reported in Figure 2, our own experiments showed 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.

2.2 Data Sampling

Approximate query processing via sampling is a popular technique [2–4, 9, 17, 18, 29] 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)* that is smaller than, and derived from, the original table(s). The sampled table(s) can be maintained by the same RDBMS. Since sampling approaches incur an additional overhead to produce the sampled tables, these are typically performed in an *offline* manner [3,9]: Once the samples are created and stored in the database, they can be interactively queried and visualized many times. (Samples can also be periodically updated when new data arrives [3].)

There is, of course, a tradeoff between output result quality and the number of samples (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 sample budget is a known issue in the approximate query processing literature [3]. Of course, in any specific real-world deployment, we expect that the application will have a fixed maximum runtime or number of samples that the system must observe.

2.3 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 who is using it, evaluating any sampling method means examining how users actually employ the visualizations they produce. Schneiderman, et al. proposed a taxonomy for information visualization types [32] 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.

2.4 Our Approach

We propose a method that proceeds in two steps: (1) during *offline preprocessing*, we produce the sample database that enables fast queries later, and (2) at *query time*, we choose samples from this database that are appropriate for the specific query.

The core innovation of our work is that we choose samples according to a *visualization-specific metric*. That is, we believe that when the samples are chosen according to the metric we propose in Section 3 below, the user will be able to accomplish her goals from Section 2.3 above (i.e., regression, density estimation, clustering) using the resulting visualization, even with a small number of rendered data points. We do *not* claim that our method will work for other visualization types, or even for other visualization goals. Indeed, our method of choosing samples could in principle be harmful for 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 declared an interest in one of these goals would be a straightforward task.

3. PROBLEM FORMULATION

The first step in our approach to obtaining a good sample for scatter plot visualizations is defining 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 In Section 4. Also, in Section 6.2, 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 with some notations. We denote a dataset D of N tuples by $D = \{t_1, t_2, \ldots, t_N\}$. Each tuple t_i is a pair of two values: (possibly multidimensional) independent variables tx_i and dependent variables ty_i. The tx_i will be a scalar value if the X-axis of a scatter plot is one-dimensional (e.g., a time series), and tx_i will be a 2D vector if the X-axis of a scatter plot is two-dimensional (e.g., the longitude/latitude in a map plot). A sample S is a subset of the dataset D, denoted by $S = \{s_1, s_2, \ldots, s_K\}$. Each s_i is a pair of sx_i and sy_i, where sx_i is the independent variable, and sy_i is the dependent one. The sample size K is pre-determined (based on the interactive latency requirements; see Section 2.2), and is given as input to our problem.

In the following, we first define a loss function to capture the visual utility for regression tasks which are one of the main visualization goals targeted in our paper (see Section 2.3). Later in Section 5 we extend our solutions to density estimation tasks, and in Section 6.2.1 we empirically show that a visualization with high regression utility is also highly effective for clustering tasks. We define our loss function (for regression) based on the intuition that users tend to rely visually on nearby points to guess or estimate the missing value of dependent variable (y) for a given value of the independent variable (x). Taking into account the nearby points for regression (a.k.a. *smoothness*) is a fundamental idea in many existing regression analyses, and kernel regression is one of the most flexible regression techniques that does not assume any underlying model [8, 10]. Based on this intuition, we use the kernel regression formulation for measuring the regression error/uncertainty at any given point, capturing its proximity to other points visualized on the plot.

Suppose we want to estimate the y value at an arbitrary point x. Given a sample S, the kernel regression such as the popular Nadaraya-Watson estimator [27, 35] is defined as:

$$\hat{y} = \frac{1}{\sum_{(sx_i, sy_i) \in S} \kappa(x, sx_i)} \sum_{(sx_i, sy_i) \in S} \kappa(x, sx_i) \cdot sy_i \qquad (1)$$

where $\hat{y}$ is the estimate of y, and $\kappa(\cdot, \cdot)$ is a function for measuring the similarity between two points. The similarity function is often called a kernel function, and a popular choice for the kernel function is the radial basis function $\kappa(x_i, x_j) = \exp(-\|x_i - x_j\|^2/\epsilon^2)$, where the parameter ϵ represents the distance that is considered as *close* by our users [36].[1] One can understand Equation (1) as a function that produces an estimate of y by computing the weighted average of the data points in S, where every point is weighted in proportion to its similarity to x.

The kernel regression framework provides a measure for the confidence of $\hat{y}$ in Equation 1. Intuitively, when there are more data points near x, our estimation confidence would be higher. More formally, the expected error or variance of $\hat{y}$ can be computed as [10]:

$$Var(\hat{y}) \propto \frac{\delta^2(x)}{\sum_{(sx_i, sy_i) \in S} \kappa(x, sx_i)}.$$

where $\delta^2(x)$ is the *true* variance of y.

Naturally, a *good* sample is one that leads to high estimation confidence (low variance) on every value on the X-axis of a scatter plot.[2] In other words, if a sample S has the lowest expected variance on $\hat{y}$ where the expectation is averaged over every possible x, we can say that the sample is the best sample for the regression task. Since the true variance of y for all x is unknown to us, we assume that $\delta^2(x)$ is equal regardless of the value of x. Then, we define our loss function as the expected variance:

$$Var(S) = E(Var(\hat{y})) = \int \frac{1}{\sum_{s_i \in S} \kappa(x, s_i)} \, dx. \qquad (2)$$

where we use s_i in place of sx_i in the kernel function $\kappa(\cdot, \cdot)$ for simplicity. In the following, we will use this simplified notation. Also, we call $Var(S)$ the *sample variance* of S. Then, with this loss function, we can obtain the best sample S by solving the following problem:

$$S = \underset{S \subseteq D;\ |S| = K}{\arg\min} \int \frac{1}{\sum_{s_i \in S} \kappa(x, s_i)} \, dx. \qquad (3)$$

Unfortunately, exact computation of this integration would be impractical, as it would require an expensive technique such as a Monte Carlo experiment with a large number of points. However, using a second-order Taylor expansion, we can obtain a slightly simpler form that enables an efficient algorithm introduced in the next section:

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

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.,

[1] In our experiments, we simply use $\epsilon \approx \max(\|x_i - x_j\|)/100$ but there is a theory on how to choose the optimal value for ϵ as the only unknown parameter [10].

[2] If the user expresses interest in only a subset of the X values, our formulation can be adjusted in a straightforward manner, i.e., by changing the summation range.

$\int \sum \kappa(x, s_i)\, dx$ 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 changing the order of integration and summation, we obtain the following optimization formulation, which we define as our Visualization-Aware Sampling (VAS) problem.

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. Given a specific kernel function such as the radial basis function mentioned earlier, we can obtain a concrete expression for $\tilde{\kappa}(s_i, s_j)$ as follows: $\exp(-\|sx_i - sx_j\|^2/2\epsilon^2)$, which is obtained after excluding constant terms that do not affect the minimization problem. We can see that this is another radial basis function whose quantity is maximal when $sx_i = sx_j$, and it decreases as the distance between the two points increases. In general, $\tilde{\kappa}(s_i, s_j)$ is another similarity function between the two data 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 kernel function directly in place of $\tilde{\kappa}(s_i, s_j)$. For convenience, we will call $\tilde{\kappa}(s_i, s_j)$ itself a kernel function as well.

Finally, note that our VAS problem has to be solved separately for every independent variable of interest. In other words, and similarly to stratified sampling [3, 9], users must declare the set of columns that they want to plot on the X-axis *a priori*, so that the VAS can be solved for each such column in an *offline* manner. However, users can choose arbitrary columns for the Y-axis at run-time, without having to declare them in advance.

While this may seem to limit the system's usability, it still covers a wide variety of real-world data analytics scenarios as certain columns tend to be used much more frequently on the X-axis. For example, *time* tends to a popular X-axis in time series data while a pair of *latitude* and *longitude* will likely be a popular choice of X-axis in a map plot. In fact, Agarwal, et al. [3] report that 90% of the queries in Conviva and Facebook can be answered using just 10% or 20% of the columns, respectively.

In the next section, we show that the VAS problem defined above is indeed NP-hard and we present an efficient approximate algorithm for solving this problem. Later in Section 6.2 we show that by finding a sample S that minimizes our loss function, we obtain a subset that, when visualized, best allows users to perform various regression tasks.

4. SOLVING VAS

In this section, we focus on solving the optimization problem derived in the previous section to obtain an optimal sample S for the regression task. We discuss an extension of our sampling process for other tasks (such as density estimation and clustering) in Section 5. First, we have the following formal result (see Appendix A for proof):

Theorem 1. VAS(Problem 1) is 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 6.4.1. Thus, in the rest of this section we present an approximation algorithm for VAS (Section 4.1), followed by further improvement ideas (Section 4.2).

4.1 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*. Hence, `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.

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 kernel function $O(K^2)$ times as there are K elements in the set, and we need to compute a kernel function for every pair of the 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 x, take an existing element s_i in S, and compute the responsibility of x in the set $S - \{s_i\} + \{x\}$. This computation takes $O(K)$ times. It is easy to see that if the responsibility of x in the set S with s_i replaced by x is smaller than the responsibility of s_i in the original set S, the replacement operation of s_i with the new data point x is a *valid replacement*. In this way, we can compare just the responsibilities without computing all pairwise kernel 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

Algorithm 1: Interchange algorithm.

input : $D = \{x_1, x_2, \ldots, x_N\}$
output: Sample S of size K which is an approximate solution to the VAS problem

```
// set for pairs of (item, responsibility)
R ← ∅
foreach x_i ∈ D do
    if |R| < K then R ← Expand (R, x_i) else
        R ← Expand (R, x_i)
        R ← Shrink (R)
    end
end
S ← pick the first item of every pair in R
return S
subroutine Expand (R, x)
    rsp ← 0      // responsibility
    foreach (s_i, r_i) ∈ R do
        l ← κ̃(x, s_i)
        r_i ← r_i + l
        rsp ← rsp + l
    end
    insert (x, rsp) into R
    return R
end
subroutine Shrink (R)
    remove (x, r) with largest r from R
    foreach (s_i, r_i) ∈ R do
        r_i ← r_i − κ̃(x, s_i)
    end
    return R
end
```

new data point x for one of the elements in S, we simply *expand* the set S by inserting x into the set, temporarily creating a set of size $K+1$. In the 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 x reduces the optimization objective of S, applying `Expand` followed by `Shrink` in Algorithm 1 replaces s_i with x. 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 $S_+ = S + \{x\}$. 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 $\tilde{\kappa}(S_- + \{x\})$, then the responsibility of the existing element s_i in an expanded set, $\text{rsp}_{S_+}(s_i)$, is also larger than the responsibility of the new element x in the expanded set, $\text{rsp}_{S_+}(x)$. The proof is as follows:

$$\begin{aligned}
&\tilde{\kappa}(S_- + \{s_i\}) > \tilde{\kappa}(S_- + \{x\}) \\
\iff &\sum_{s_j \in S_-} \tilde{\kappa}(s_i, s_j) > \sum_{s_j \in S_-} \tilde{\kappa}(x, s_j) \\
\iff &\tilde{\kappa}(s_i, x) + \sum_{s_j \in S_-} \tilde{\kappa}(s_i, s_j) \\
&\qquad > \tilde{\kappa}(s_i, x) + \sum_{s_j \in S_-} \tilde{\kappa}(x, s_j) \\
\iff &\text{rsp}_{S_+}(s_i) > \text{rsp}_{S_+}(x).
\end{aligned}$$

Since the responsibility of s_i is larger than that of x in the expanded set S_+, the `Shrink` routine will remove s_i. If no element exists whose responsibility is larger than that of x, then x 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 set obtained by `Interchange` is S_{int}, and the optimal sample set is S_{opt}. The quality of S_{int}, or the optimization objective, has the following upper bound:

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

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

Proof. Due to the submodularity of VAS, which we show in Appendix B, we can apply the result of Nemhauser, et al. [28] and obtain the result above. □

Ideally, `Interchange` should run until no valid replacement is available, but in practice, we observed that a single scan over the dataset produces a sample of good quality if the dataset is shuffled. In cases where the data is in order, we performed three scans over the dataset.

4.2 Further Speed-Up

The time complexity of `Interchange` is $O(N \cdot K)$, where N is the original dataset size and K is the sample size we want to obtain. Thus, we found that the runtime is still slow, especially for samples of large sizes, e.g., $K = 1$M and $N = 1$B points. In this section, we describe an idea to further speed up the algorithm at the cost of small computational error.

Kernel Locality — Similarity kernels such as radial basis kernels have a property called *locality*. The locality property of a kernel function indicates that its value becomes negligible when the distance between the two data points

is not very close—an idea also used in accelerating previous algorithms [20]. For example, for the radial basis kernel, $\exp(-\|x_i - x_j\|^2/\epsilon^2)$, the 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 quantity, it will not affect the final outcome very much. Using this property, we can make the `Expand` and `Shrink` operations much faster than before by only considering data points in the current sample that are close enough to a new data point.

Implementation Details — We use the following approach to make the Expand and Shrink operations exploit kernel locality more quickly. Because checking every data point in a sample to identify the data points close to a new data point incurs the same amount of time complexity, we create a grid with its cell width equal to ϵ. Next, we always keep all the sample data in the grid so that when a new data point is considered for a valid replacement, we can update just the responsibilities of the data points in the cells near the new data point. To quickly identify the data points with the largest responsibility in the `Shrink` operation, we maintain a max-heap. With this approach using the kernel locality, we can speed up `Interchange` more than 50 times when the sample size is 10,000 or larger.

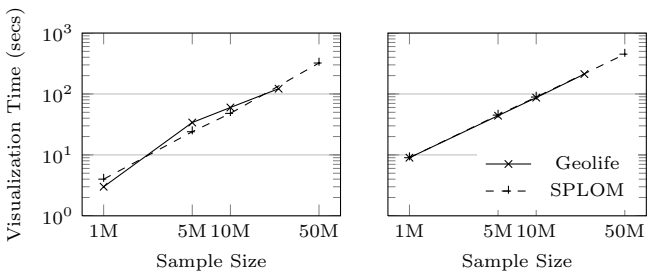
5. EMBEDDING DENSITY

In Section 3, we presented our VAS formulation for a visualization with regression goals, requiring us to minimize the sample variance in Equation (2).

This sampling approach, however, does not necessarily ensure high utility for other visualization goals that we may be interested in, such as density estimation and clustering. Note that the core idea of density estimation and clustering is to understand the distribution of data points in the original dataset, but obtaining a sample that represents the data distribution of the original dataset is in conflict with obtaining a good sample for regression. This is because a sample optimized for the regression task tends to have data points spread well across the domain, as visualized in Figure 1d, which represents a sample of data points obtained without reference to the distribution of the original dataset.

In order to arbitrate these conflicting goals, we propose an extension to VAS in this section. Our proposed extension is embedding density information to the sample created by VAS as a post-processing step. More concretely, our proposed approach works as follows: (1) obtain a sample using our algorithm for VAS, and then (2) while scanning the dataset once more, increase the counter for a sampled data point if it is the nearest neighbor of the data point just scanned. With this extra counter for every sampled data point, we can now see the density of the data points in the sample and also visualize the information, e.g., using different dot sizes or by adding jitter noise in proportion to each point's density. (See Section 6.2 for an example.)

Interestingly, we can also show that this approach is the *optimal* way of displaying the density information by considering that this task is yet another regression task in which the dependent variable y now represents density. Thus, in this situation, we can apply exactly the same logic as in Section 3 and still be assured that embedding density in the sample obtained by VAS is the optimal way of showing the density information.



(a) Tableau (b) MathGL

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

In the second stage of the aforementioned process for embedding density, a special data structure such as a k-d tree [7] can be used to make the nearest neighbor tests more efficient. This can be 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 $O(N \log K)$. Thus, the overall time complexity of VAS remains unchanged, even after being augmented with this density estimation/encoding step.

6. EXPERIMENTS

We ran four types of experiments to demonstrate that VAS and VAS with density embedding can produce better plots in less time than competing methods.

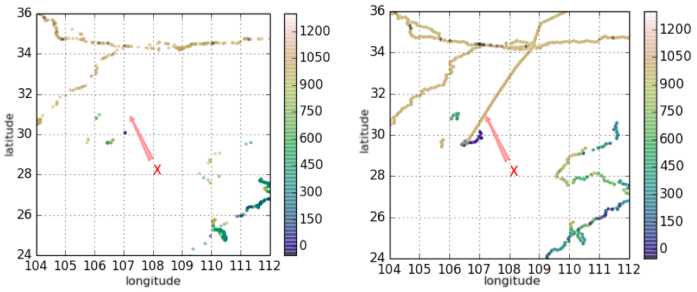
1. We continue the runtime study of existing visualization systems that we introduced in Figure 2.

2. In a user study we show that users are more successful when they use visualizations produced by VAS than with other methods. We also show that user success and our loss function are strongly negatively correlated (that is, users are successful when are loss function is minimized).

3. We show that VAS can obtain a sample of a fixed quality level (that is, loss function level) with fewer samples than competing methods. We demonstrate this over a range of different datasets and sample quality levels.

4. We examine two ramifications of our design choices for the algorithm: the VAS offline sample time and runtime costs of computing an exact vs. an approximate solution.

All of our experiments were performed using two datasets. The Geolife dataset was collected by Microsoft Research [38–40]. It contains latitude, longitude, elevation triples from GPS loggers, recorded mainly around Beijing. Our full database contains 24.4M tuples. We also used SPLOM, a synthetic dataset generated from several Gaussian distributions that has been used in previous visualization projects [19, 23]. We used parameters identical to previous work, and generated a dataset of five columns and 1B tuples. We evaluated all runtimes on machines with 16 CPUs and 122GB of RAM.



(a) Stratified Sampling (b) VAS

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

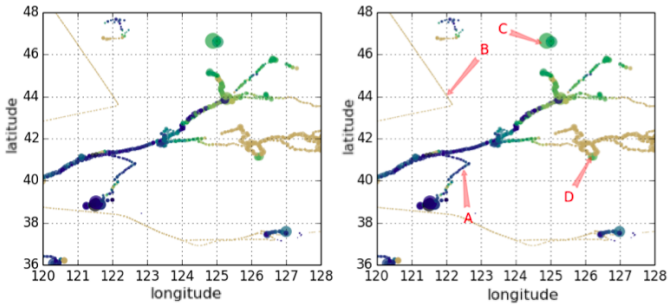


Figure 6: An example figure used in the user study for the density estimation task. This figure is generated using VAS with density embedding. The left-hand image is the original figure. The right-hand image contains four test markers, used to ask users to choose the densest area and the sparsest areas.

6.1 Existing Systems are Slow

Our discussions in this paper are based on the idea that plotting a scatter plot using an original dataset takes a unacceptably long time. We tested two state-of-the-art systems: Tableau [34] and MathGL [25]. 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 the time to load data from SSD storage and the time it takes 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 grows linearly with sample size.

6.2 User Success and Sample Quality

In this section we make two important experimental claims about user interaction with visualizations produced by our system. First, that 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, that user success and our loss function — that is, our measure of sample quality — are correlated. We validate these claims with a user study performed using Amazon's Mechanical Turk system.

6.2.1 *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 him 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. We thus 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.

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 discretizes the 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 has only 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 it 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 1a 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 of sizes 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 zoom-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 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 1b. Interestingly, the basic VAS method *without* density

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

Table 1: User performance in the regression, density estimation, and clustering tasks. Uniform is uniform random sampling, Stratified is stratified sampling, and VAS is VAS. "VAS w/ density" is VAS with the density information added to the sample.

estimation yielded very poor results. As noted earlier in Section 5, VAS ignores the underlying density of points in the original dataset. 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 have 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 does not have ground-truth for clustering, we used instead synthetic datasets that we generated using Gaussian distributions. Using two-dimensional Gaussians with different covariances, we generated 4 datasets, 2 among 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 of sizes 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 Turkers and filtered out bad workers.

Table 1c 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 are generated from two Guassian distributions, the data points sampled by VAS 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 performs 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 are more clusters than actually exist.

6.2.2 Correlation with Sample Quality

In this section, we test whether the VAS optimization criterion of sample variance — our measure of sample quality — the VAS loss function — has a close relationship with our visualization users' success in reaching their end goals. If they are highly correlated, we have some empirical evidence that samples which minimize the VAS loss function will yield useful plots.

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 the sample variance using the expression in Equation 2. We then measured the correlation between the sample variance and average user performance on the regression task for that visualization.

To compute the sample variance, 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 considered randomly generated points are within the domain if there exists any data point in the original dataset whose distance to the randomly generated data points is no larger than 0.1. Now, the integral expression is replaced with a summation as follows:

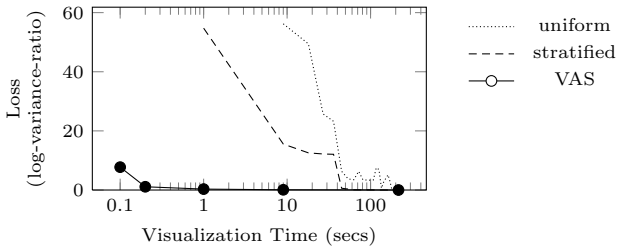
$$Var(S) = \frac{1}{1000} \sum_{i=1}^{1000} \frac{1}{\sum_{s_i \in S} \kappa(x_i, s_i)}.$$

This sample variance computed as above is the *mean* of one thousand variances. One problem we encountered in computing the mean was that the variances often became so large that `double` precision could not hold those quantities. To address this, we used the *median* in place of the *mean* in this section because the median is less sensitive to a few extreme values. 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. In the following sections, we report both the mean and the median of samples.

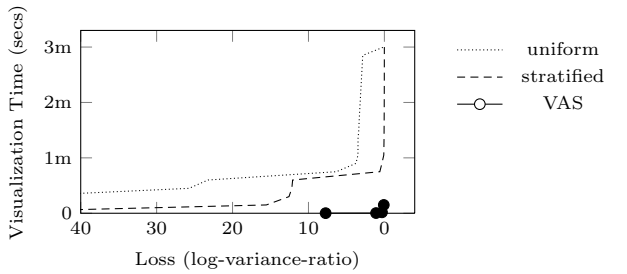
Next, to examine the sample variances in a more consistent way, we computed the following quantity:

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

where D is the original dataset. $Var(D)$ is the lowest sample variance that a sample can achieve; thus, samples with log-variance-ratios close to zero can be regarded as good samples according to our loss function.



(a) Loss (quality) for a given visualization time. Smaller loss is better.



(b) Visualization time for a fixed loss (quality) value.

Figure 7: Relationship between visualization production time and loss function (quality) for the three sampling methods.

Next we examined the relationship between a sample's log-variance-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. For this, we computed Spearman's rank correlation coefficient, which indicates -1.0 for rankings that are perfectly negatively correlated, 1.0 for rankings that are perfectly positively correlated, and a value close to 0 for rankings that are uncorrelated. When we computed the correlation coefficient from the data we obtained from the user study, we obtained a coefficient of -0.85, indicating a strong negative correlation between user success and the log variance ratio. (Its p-value was 5.2×10^{-4}. The data used for the correlation calculation is plotted in Appendix D.1.) 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 3 and the intuition behind it was largely valid.

6.3 VAS Uses Fewer Samples

This section shows that VAS can produce better samples 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 samples.)

We used the Geolife dataset and produced samples of various sizes (and thus, different visualization production times). Figure 7a shows the results when we vary the visualization time: VAS always produces samples with lower loss function values (i.e., higher quality) than other methods. (This diagram shows the mean log-variance-ratio; a similar diagram using median can be found in Appendix D.2.) The quality gap between the methods does not become small until after an entire minute of runtime. We show a similar result with another dataset in Appendix D.2.

Figure 7b 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. (Smaller loss values are better, and so are depicted on the right side of the graph.) Because the samples generated by our method have much smaller losses compared to other methods, all of the data points in the figure are in the right bottom corner. Competing methods required much more time than VAS to obtain the same quality (loss function value).

Of course one could always improve the performance of stratified sampling by carefully adjusting the number of its bins (which was 100 in our experiments) based on the sample size and the underlying data distribution. However, finding the optimal number of bins for a large dataset is an involved and expensive process (compared to VAS which only requires a single parameter ϵ). Moreover, note that VAS is still provably optimal for minimizing our loss function which we have shown to be strongly correlated with user success (see Section 6.2.2).

6.4 Algorithmic Details

We now examine two internal qualities of the VAS technique: approximate vs exact solution, and its offline sampling time.

6.4.1 *Exact vs. Approximate*

Appendix A, which shows the NP-hardness of VAS, supports the need for an approximate algorithm. This section empirically examines the NP-hardness of VAS when the radial basis function, $\exp(-\|s_i - s_j\|^2/\epsilon^2)$, is used for the kernel function.

We think one of the best options for obtainiing 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 Appendix C for converting VAS to an instance of Mixed Integer Programming (MIP). We used the GNU Linear Programming Kit [24] to solve the converted MIP problem.

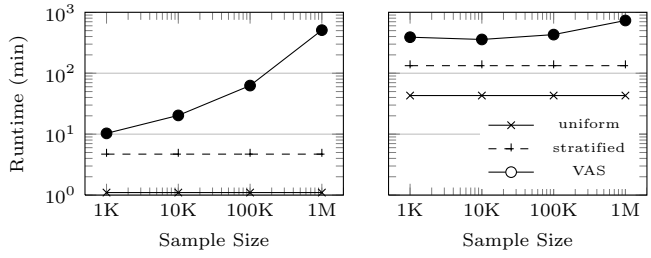
Table 2 shows the time it takes to obtain exact solutions to VAS with datasets of very small sizes. The sample size K was fixed to 10 in all of the experiments in the table. According to the result, the exact solutions to VAS showed better quality, but the procedure to obtain them took considerably longer. 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 for any but extremely small data sizes.

6.4.2 *Offline Sampling Time*

In the offline phrase, the VAS method is far more expensive than competing approaches: is it fast enough to remain feasible? We measured the runtime of `Interchange`, changing the sample sizes from 1K to 1M. The results are shown in Table 8. At its most expensive, `Interchange` took approximately 12 hours to produce a sample of 1M records from

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

Table 2: Variance and runtime comparison of the algorithms for VAS. Uniform random sampling (Random) is included for reference. MIP stands for Mixed Integer Programming, a procedure for obtaining exact solutions to VAS. K is fixed to 10 in all the experiments above.



(a) Geolife (b) SPLOM

Figure 8: Offline sampling time for the three sampling methods. VAS offline sampling is time-consuming, but can produce a sample of 1 million data points from an original dataset of 1B points in a reasonable amount of time.

the SPLOM database of 1B records. This offline phase is not fast, but is not unreasonable: it is on the same rough time scale as large-scale database indexing tasks. For many (perhaps most) applications, optimizing the query-time user experience will be worth a large amount of offline work.

7. RELATED WORK

Support for interactive visualization of large datasets is a fast-growing area of research interest [5, 6, 11, 12, 14–16, 21, 23, 30, 37]. 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. [6]. 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.

Binned aggregation approaches [21, 23, 37] 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 [37] 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 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 and thus makes smoothing faster. The *abstract rendering pipeline* [12] 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 [11, 30]. It is helpful but largely orthogonal to our contributions.

Incremental visualization proposes a streaming data processing model, which quickly yields an initial low-resolution version of the user's desired bitmap [14, 15]. 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 2, 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.

8. 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 further showed that the VAS offline phase, while more time-consuming than competitors', is still feasible for large databases.

We envision two areas for extending this work in the future. First, we want to research some of the connections between our proposed algorithm and other domains not directly related to visualization, such as optimal sensor placement [20]. Second, we believe our core topic — data system support for visualization tools — is still in its infancy and contains a range of interesting challenges. In particular, we would like to investigate techniques for rapidly generating visualizations for other user goals (including outlier detection, trend identification, and others) and other visualization types (such as large networks).

9. REFERENCES

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

[2] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD*, 1999.
[3] 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.
[4] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.
[5] 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.
[6] L. Battle, M. Stonebraker, and R. Chang. Dynamic reduction of query result sets for interactive visualizaton. In *BigData Conference*, pages 1–8, 2013.
[7] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of ACM*, 18, 1975.
[8] C. M. Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.
[9] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 2007.
[10] P. Cizek, W. K. Härdle, and R. Weron. *Statistical tools for finance and insurance*. Springer, 2005.
[11] J. A. Cottam and A. Lumsdaine. Automatic application of the data-state model in data-flow contexts. In *IV*, pages 5–10, 2010.
[12] J. A. Cottam, A. Lumsdaine, and P. Wang. Overplotting: Unified solutions under abstract rendering. In *BigData Conference*, 2013.
[13] U. Feige, D. Peleg, and G. Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29, 2001.
[14] D. Fisher, S. M. Drucker, and A. C. König. Exploratory visualization involving incremental, approximate database queries and uncertainty. *IEEE Computer Graphics and Applications*, 32(4):55–62, 2012.
[15] 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.
[16] J. Heer and S. Kandel. Interactive analysis of big data. *XRDS*, 2012.
[17] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. *SIGMOD*, 1997.
[18] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the dbo engine. *TODS*, 2008.
[19] 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.
[20] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *JMLR*, 9, 2008.
[21] L. D. Lins, J. T. Klosowski, and C. E. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *TVCG*, 2013.
[22] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. 2002.
[23] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, 2013.
[24] A. Makhorin. Glpk (gnu linear programming kit), version 4.54. `http://www.gnu.org/software/glpk`.
[25] Mathgl. `http://mathgl.sourceforge.net/doc_en/Main.html`.
[26] R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, 1968.
[27] E. Nadaraya. On estimating regression. *Theory of Probability And Its Applications*, 9(1):141–142, 1964.
[28] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14, 1978.
[29] C. Olston, E. Bortnikov, K. Elmeleegy, F. Junqueira, and B. Reed. Interactive analysis of web-scale data. In *CIDR*, 2009.
[30] H. Piringer, C. Tominski, P. Muigg, and W. Berger. A multi-threading architecture to support interactive visual exploration. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1113–1120, 2009.
[31] B. Shneiderman. Response time and display rate in human performance with computers. *CSUR*, 1984.
[32] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages*, 1996.
[33] I. Stoica. For big data, moore's law means better decisions. `http://www.tableausoftware.com/`.
[34] Tableau software. `http://www.tableausoftware.com/`.
[35] G. S. Watson. Smooth regression analysis. *SankhyĂĄ: The Indian Journal of Statistics, Series A (1961-2002)*, 26(4):359–327, 1964.
[36] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2009.
[37] H. Wickham. Bin-summarise-smooth: a framework for visualising large data. Technical report, had.co.nz, 2013.
[38] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma. Understanding mobility based on gps data. In *Ubiquitous computing*, 2008.
[39] Y. Zheng, X. Xie, and W.-Y. Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 2010.
[40] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *WWW*, 2009.

APPENDIX

A. HARDNESS OF VAS

In this section, we prove the NP-hardness of VAS under the condition that the kernel function $\tilde{\kappa}(s_i, s_j)$ can have any arbitrary value, which is true in a general setting. One can argue that, however, in practice, x_i's are enough to be defined in a metric space in which triangular inequality must hold, so the values of the kernel functions should retain related property. To address the question, we conduct empirical study in Section 6 with Mixed Integer Programming toolkit to directly solve the problem. The result of the emprical study also supports our claim.

Theorem 4. 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 [13].

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 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. □

B. SUBMODULARITY OF VAS

Here, we show that VAS has the property called *submodularity*. The property helps us provide a worst-case bound for the approximate algorithm that will be presented in the following section. Although the wost-case bound is interesting in a theoretical perspective, the approximate algorithm seems to create samples whose qualities are quite close to those of optimal samples according to our empirical study in Section 6.4.2.

Before showing the submodularity of VAS, we first introduce the definition of submodularity.

Definition 3. (Submodularity) A real-valued function f defined over subsets of a dataset D is *submodular* if for all $S \subseteq T \subseteq D$ and all $x_i \in D - T$, f has the following property:

$$f(S \cup \{x_i\}) - f(S) \geq f(T \cup \{x_i\}) - f(T)$$

The intuitive interpretation of the submodularity is that the gain of adding a new element x_i to an existing set decreases as the size of the set increases. To show the submodularity of the problem, we formulate a variant of the problem that returns the same answer as follows:

$$S = \underset{S \subseteq D;\ |S|=K}{\arg\max} \sum_{s_i,s_j \in S;\ i<j} 1 - \tilde{\kappa}(s_i, s_j) \tag{4}$$

where we assume that the maximal value of $\tilde{\kappa}(\cdot, \cdot)$ is 1, which is the condition that can be easily met by normalization.

Lemma 2. Let $f(S) = \sum_{s_i,s_j \in S;\ i<j} (1 - \tilde{\kappa}(s_i, s_j))$, then the function f is submodular.

Proof. Let $S \subseteq T \subseteq D$. $f(S \cup \{x\}) - f(S) = \sum_{s_i \in S} (1 - \tilde{\kappa}(x, s_i))$, and $f(T \cup \{x\}) - f(T) = \sum_{s_i \in T} (1 - \tilde{\kappa}(x, s_i))$. Then

$$\begin{aligned}&(f(T \cup \{x\}) - f(T)) - (f(S \cup \{x\}) - f(S)) \\ &= \sum_{s_i \in T - S} (1 - \tilde{\kappa}(x, s_i)) \geq 0\end{aligned}$$

since $\tilde{\kappa}(\cdot, \cdot) \leq 1$. □

C. MIP FORMULATION

In order to find an exact solution to Max Cover Sampling (MCS), we can covert the problem to an instance of mixed integer programming. The mixed integer programming is a constrained version of linear programming in which some of variables should be one of predefined integer values. The problem of MCS can be expressed as follows:

$$\begin{aligned}\min \quad & \sum_{\substack{i,j \in [1,N],\\ i<j}} \exp(-c \cdot d_{ij}) \cdot b_{ij} \\ \text{s.t.} \quad & \sum_{i \in [1,N]} a_i = K, \quad b_{ij} = a_i \cdot a_j, \quad a_i, b_{ij} \in \{0, 1\}\end{aligned}$$

where N is the dataset cardinality, and a_i indicates x_i is chosen. The above problem is not yet *linear* because it contains multiplication in its constraints. Therefore, to solve the problem using any toolkit for MIP, we should linearize the problem.

Note that since both a_i and b_{ij} are binary variables in the above problem, we can think $b_{ij} = a_i$ AND a_j. It is known that we can convert the logical AND operation to the following equivalent expression for integer programming:

$$\begin{aligned}&b_{ij} \leq a_i \\ &b_{ij} \leq a_j \\ &b_{ij} \geq a_i + a_j - 1 \\ &b_{ij} \geq 0\end{aligned}$$

Using the above relationship, now we can convert the above problem to the following one that can be consumed by a MIP toolkit such as GLPK:

$$\begin{aligned}\min \quad & \sum_{\substack{i,j \in [1,N],\\ i<j}} \exp(-c \cdot d_{ij}) \cdot b_{ij} \\ \text{s.t.} \quad & \sum_{i \in [1,N]} a_i = K, \quad a_i, b_{ij} \in \{0, 1\} \\ & b_{ij} \leq a_i, \quad b_{ij} \leq a_j, \quad b_{ij} \geq a_i + a_j - 1, \quad b_{ij} \geq 0\end{aligned}$$

With the above ILP formulation, we can apply existing approaches to ILP problems, and the obtained solution is optimal. The solution obtained by solving this ILP problem has minimal loss, and can be applied when N is very small, say less than 100. When dataset is of size of interest, however, it is not a tractable approach.

D. MORE EXPERIMENTS

D.1 Correlation between Sample Variance and User Success Ratio

Figure 9 depicts the negative correlation bewteen the sample variance and user success ratio in the regression task. Each dot in the figure is obtained by averaging the the fraction of the correct answers submitted by 40 unique Turkers on 6 questions. Because the X-axis of the figure is loss function we aim to minize to obtain a good sample, the negative correlation bewteen the two metrics shows the validity of our problem formulation.

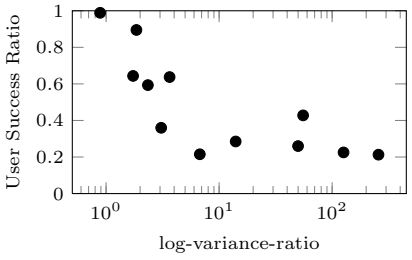
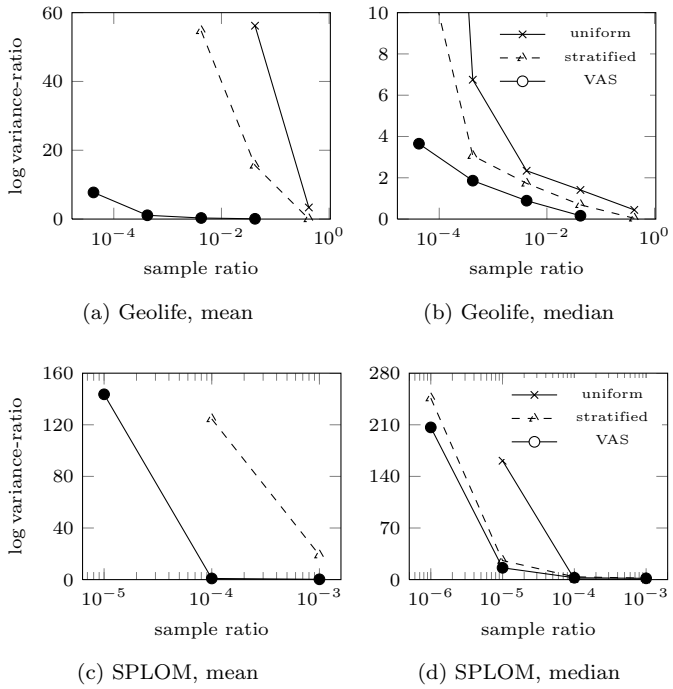


Figure 9: The relationship between the sample variance and user performance on the regression task. The samples with smaller sample variances resulted in better success ratios in general in the regression task.

D.2 More Results on Sample Variance Comparison

We report both the mean and median of log-variance-ratios introduced in Section 6.2.2 for two datasets (Geolife



(a) Geolife, mean (b) Geolife, median

(c) SPLOM, mean (d) SPLOM, median

Figure 10: The sample variances of different sampling methods with different sample sizes. The sample variances were computed in two different ways: mean and median. We used two datasets (Geolife and SPLOM). We did not include the points whose sample variances were beyond the limit of `double` type.

and SPLOM). See Figure 10. The X-axis of the plot shows the ratio of the size of the samples we obtained relative to the original dataset, and the Y-axis shows the loss function we used. Therefore, the lower the better.

Due to some extreme quantities that appear in calculating the sample variance using Monte Carlo technique, there is a large gap between our method and the other sampling methods. To avoid the skewness of those sample variance computations, we also included median graphs for comparison. Note that in the graphs, because the Y-axis measures the log value of the variance-ratio, the difference of 1 indicates a quality difference of 10 times. In all cases, the samples by our method achieved much lower sample variances, and the gap was bigger when the sample sizes were small. (This is natural because if the sample ratio is 1, there should not be any quality difference.)

D.3 OpenStreetMap Data Experiment

OpenStreetMap dataset contains 9 years of GPS traces uploaded to OpenStreeMap project by individuals, and we obtained the dataset from its repository.[3]. We extracted from the data the GPS logs that contained altitude or elevation record together with a pair of latitude and longitude. We processed raw `xml` files, and it took more than a day to extract all the fields into a more condensed `csv` file. The total number of the records we extracted from the dataset was 2,195,899,509, and the size of the extracted `csv` file was 59G on disk.

Figure 11 reports the time it took to process the data by various sampling methods. The sample sizes range from 1K

[3]http://planet.openstreetmap.org/gps/

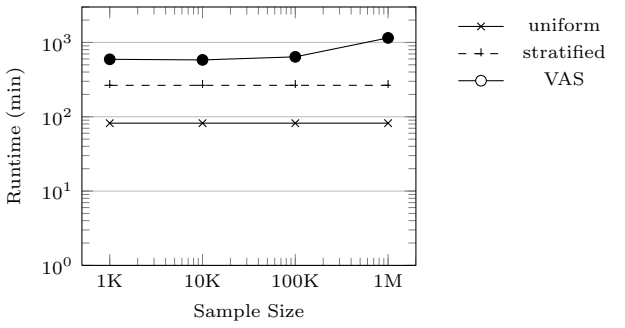
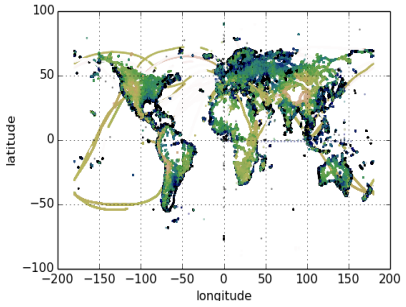


Figure 11: Offline sampling time of VAS. This dataset contains more than 2B tuples.

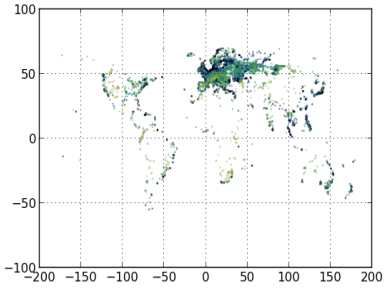
to 1M, and it took about 1,100 mins for VAS to process the entire dataset to obtain a sample of 1M records. Notably, due to the large size of the dataset, the sample speed was not different as much as the experiments with the Geolife dataset.

For comparison purpose, we generated several scatter plots into image files, and show them in Figure 12. The first figure is a scatter plot we obtain by processing 25% of the entire dataset. The 25% of the dataset was the limit we could process using the `pyplot` library on a machine with 244G memory. Also, generating the plot took huge amount of time. For reading a dataset from disk and parsing, it took 49 minutes, and for rendering the loaded data into a scatter plot took more than 4 hours. See that the visualization time only for 25% of the data took 4 hours, and this indicates that it will take possible more than 10 hours for visualization of the entire dataset, if memory allows. This indicates that when the dataset is massive, sampling-based visualization will be much more appealing.

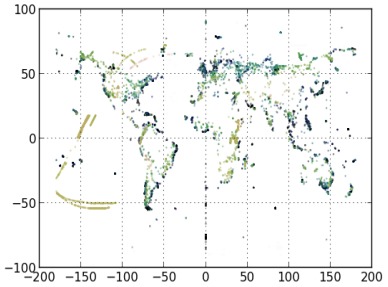
Figure 12d is a scatter plot generated by processing only 10K records. Despite its tiny sample size compared to the original dataset (5×10^{-8}%), the figure visualizes many interesting areas that were missed by other sampling methods in Figure 12b and c. Certainly, it we zoom into a particular area, we need to use samples with more records to display rich information. However, at least, this example shows that the high quality of the scatter plot generated by VAS.



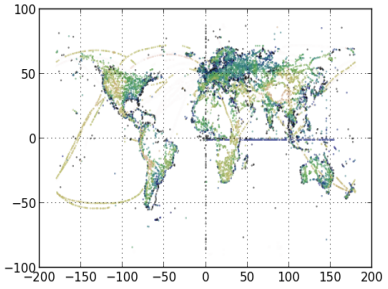
(a) 25% of the original 2B tuple dataset



(b) A 10K tuple random sample



(c) A 10K tuple stratified sample



(d) A 10K tuple VAS sample

Figure 12: 25% of the original OpenStreetMap dataset and samples of 10K data points.