I've Seen "Enough": Incrementally Improving Visualizations to Support Rapid Decision Making

Sajjadur Rahman[1] Maryam Aliakbarpour[2] Ha Kyung Kong[1] Eric Blais[3] Karrie Karahalios[1,4]
Aditya Parameswaran[1] Ronitt Rubinfield[2]
[1]University of Illinois (UIUC) [4]Adobe Research [2]MIT [3]University of Waterloo
{srahman7,hkong6,kkarahal,adityagp}@illinois.edu {maryama@,ronitt@csail}.mit.edu eblais@uwaterloo.ca

ABSTRACT

Data visualization is an effective mechanism for identifying trends, insights, and anomalies in data. On large datasets, however, generating visualizations can take a long time, *delaying the extraction of insights, hampering decision making, and reducing exploration time*. One solution is to use online sampling-based schemes to generate visualizations faster while improving the displayed estimates incrementally, eventually converging to the exact visualization computed on the entire data. However, the intermediate visualizations are approximate, and often fluctuate drastically, leading to potentially incorrect decisions. We propose sampling-based incremental visualization algorithms that reveal the "salient" features of the visualization quickly—with a $46\times$ speedup relative to baselines—while minimizing error, thus enabling rapid and error-free decision making. We demonstrate that these algorithms are *optimal* in terms of sample complexity, in that given the level of interactivity, they generate approximations that take as few samples as possible. We have developed the algorithms in the context of an incremental visualization tool, titled INCVISAGE, for trendline and heatmap visualizations. We evaluate the usability of INCVISAGE via user studies and demonstrate that users are able to make effective decisions with incrementally improving visualizations, especially compared to vanilla online-sampling based schemes.

1. INTRODUCTION

Visualization is emerging as the most common mechanism for exploring and extracting value from datasets by novice and expert analysts alike. This is evidenced by the huge popularity of data visualization tools such as Microsoft's PowerBI and Tableau, both of which have 100s of millions of dollars in revenue just this year [5, 3]. And yet data visualization on increasingly large datasets, remains *cumbersome*: when datasets are large, generating visualizations can take hours, *impeding interaction, preventing exploration, and delaying the extraction of insights* [36]. One approach to generating visualizations faster is to sample a small number of datapoints from the dataset online; by using sampling, we can view visualizations that incrementally improve over time and eventually converge to the visualization computed on the entire data. However, such intermediate visualizations are approximate, and often fluctuate drastically, leading to *incorrect insights and conclusions*. The key question we wish to address in this paper is the following: ***can we develop a sampling-based incremental visualization algorithm that reveals the features of the eventual visualization quickly, but does so in a manner that is guaranteed to be correct?***

Illustrative Example. We describe the goals of our sampling algorithms via a pictorial example. In Figure 1, we depict, in the first row, the variation of present sampling algorithms as time progresses and more samples are taken: at t_1, t_2, t_4, t_7, and when all of the data has been sampled. This is, for example, what visualizing the results of an online-aggregation-like [19] sampling algorithm might provide. If a user sees the visualization at any of the intermediate time points, they may make incorrect decisions. For example, at time t_1, the user may reach an incorrect conclusion that the values at the start and the end are lower than most of the trend, while in fact, the opposite is true—this anomaly is due to the skewed samples that were drawn to reach t_1. The visualization continues to vary at t_2, t_4, and t_7, with values fluctuating randomly based on the samples that were drawn. Indeed, a user may end up having to wait until the values stabilize, and even then may not be able to fully trust the results. One approach to ameliorate this issue would be to use confidence intervals to guide users in deciding when to draw conclusions—however, prior work has demonstrated that users are not able to interpret confidence intervals correctly [14]. At the same time, the users are subject to the same vagaries of the samples that were drawn.

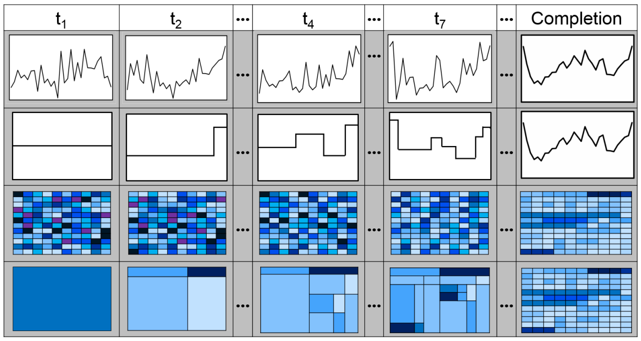


Figure 1: INCVISAGE **example.**

Another approach, titled INCVISAGE, that we espouse in this paper and depict in the second row is the following: at each time point t_i, reveal one additional segment for a i-segment trendline, by splitting one of the segments for the trendline at t_{i-1}, when the tool is confident enough to do so. Thus, INCVISAGE is very conservative at t_1 and just provides a mean value for the entire range, then at t_2, it splits the single segment into two segments, indicating that the trend increases towards the end. Overall, by t_7, the tool has indicated many of the important features of the trend: it starts off high, has a bump in the middle, and then increases towards the end. This approach reveals features of the eventual visualization in the order of prominence, allowing users to gain early insights and draw conclusions early. This approach is reminiscent of interlaced pixel-based image generation in browsers [38], where the image slowly goes from being blurry to sharp over the course of the rendering, displaying the most salient features of the visualization before the less important features. Similar ideas have also been developed in other domains such as signal processing [47] and geo maps [13].

So far, we've described trendlines; the INCVISAGE approach can be applied to heatmap visualizations as well—depicted in row 4 for the corresponding online-aggregation-like approach shown in row 3—as is typical in heatmaps, the higher the value, the darker the color. Once again row 3—the status quo—fluctuates tremendously, not letting analysts draw meaningful insights early and confidently. On the other hand, row 4—the INCVISAGE approach—repeatedly subdivides a block into four blocks when it is confident enough to do so, emphasizing early, that the right hand top corner has a higher value, while the values right below it are somewhat lower. In fact, the intermediate visualizations may be preferable because users can get the big picture view without being influenced by noise.

Open Questions. Naturally, developing INCVISAGE brings a whole host of open questions, that span the spectrum from usability to algorithmic process. First, it is not clear at what rate we should be displaying the results of the incremental visualization algorithm. When can we be sure that we know enough to show the ith increment, given that the $(i-1)$th increment has been shown already? How should the ith increment differ from the $(i-1)$th increment? How do we prioritize sampling to ensure that we get to the ith increment as soon as possible, but with guarantees? Can we show that our algorithm is in a sense 'optimal', in that it aims to take as few samples as possible to display the ith increment with guarantees? And at the same time, how do we ensure that our algorithm is lightweight enough that computation doesn't become a bottleneck? How do we place the control in the user's hands in order to control the level of interactivity needed?

The other open questions involved are related to how users interpret incremental visualizations. Can users understand and make sense of the guarantees provided? Can they use these guarantees to make well-informed decisions and terminate early without waiting for the entire visualization to be generated?

Contributions. In this paper, we address all of these open questions. Our primary contribution in the paper is the notion of *incrementally improving visualizations that surface important features as they are determined with high confidence* — bringing a concept that is commonly used in other settings, e.g., rasterization and signal processing, to visualizations. Given a user specified interactivity threshold (described later), we develop incremental visualizations algorithms for INCVISAGE that minimizes error. We introduce the concept of *improvement potential* to help us pick the right improvement per increment. We find, somewhat surprisingly, that a remarkably simple algorithm works best under a *sub-Gaussian assumption* [42] about the data, which is reasonable to assume in real-world datasets (as we show in this paper). We further demonstrate that these algorithms are *optimal* in that they generate approximations within some error bound given the interactivity threshold. When users don't provide their desired interactivity threshold, we can pick appropriate parameters that help them best navigate the tradeoff between error and latency. We additionally perform user studies to evaluate the usability of an incremental visualization interface, and evaluate whether users are able to make effective decisions with incrementally improving visualizations. We found that they are able to effectively determine when to stop the visualization and make a decision, trading off latency and error, especially when compared to traditional online sampling schemes.

Prior Work. Our work is complementary to other work in the incremental visualization space. In particular, sampleAction [16] and online aggregation [19] both perform online sampling to depict aggregate values, along with confidence-interval style estimates to depict the uncertainty in the current aggregates. In both cases, however, these approaches prevent users from getting early insights since they need to wait for the values to stabilize. As we will discuss later, our approach can be used in tandem with online aggregation-based approaches. IFOCUS [30], PFunk-H [11], and ExploreSample [50] are other approximate visualization algorithms targeted at generating visualizations rapidly while preserving perceptual insights. IFOCUS emphasizes the preservation of pairwise ordering of bars in a bar chart, as opposed to the actual values; PFunk-H uses perceptual functions from graphical perception research to terminate visualization generation early; ExploreSample approximates scatterplots, ensuring that overall distributions and outliers are preserved. An early paper by Hellerstein et al. [20] proposes a similar technique of progressive rendering for scatterplot visualizations by using index statistics to depict estimates of density before the data records are actually fetched. Lastly, M4 [27] uses rasterization to reduce the dimensionality of a time series without impacting the resulting visualization. None of these methods emphasize revealing features of visualizations incrementally.

Outline. The outline of the remainder of this paper is as follows: in Section 2 we formally define the incremental visualization problem. Section 3 outlines our incremental visualization algorithm while Section 4 details the system architecture of INCVISAGE. In Section 5 we summarize the experimental results and the key takeaways. Then we present the user study design and outcomes in Section 6 (for usability) and 7 (for comparison to traditional online sampling schemes). Section 8 describes other related work on data visualization and analytics.

2. PROBLEM FORMULATION

In this section, we first describe the data model, and the visualization types we focus on. We then formally define the problem.

2.1 Visualizations and Queries

Data and Query Setting. We operate on a dataset consisting of a single large relational table R. Our approach also generalizes to multiple tables obeying a star or a snowflake schemata but we focus on the single table case for ease of presentation. As in a traditional OLAP setting, we assume that there are dimension attributes and measure attributes—dimension attributes are typically used as group-by attributes in aggregate queries, while measure attributes are the ones typically being aggregated. For example, in a product sales scenario, day of the year would be a typical dimension attribute, while the sales would be a typical measure attribute.

INCVISAGE supports two kinds of visualizations: trendlines and heatmaps. These two types of visualizations can be translated to aggregate queries Q_T and Q_H respectively:

$$\begin{aligned} \mathtt{Q_T} &= \mathtt{SELECT\ X_a, AVG(Y)\ FROM\ R} \\ &\quad \mathtt{GROUP\ BY\ X_a\ ORDER\ BY\ X_a}, \quad \text{and} \\ \mathtt{Q_H} &= \mathtt{SELECT\ X_a, X_b, AVG(Y)\ FROM\ R} \\ &\quad \mathtt{GROUP\ BY\ X_a, X_b\ ORDER\ BY\ X_a, X_b} \end{aligned}$$

Here, X_a and X_b are dimension attributes in R, while Y is a measure attribute being aggregated. The trendline and heatmap visualizations are depicted in Figure 1. For trendlines (query Q_T), the attribute X_a is depicted along the x-axis while the aggregate AVG(Y) is depicted along the y-axis. On the other hand, for heatmaps (query Q_H) the two attributes X_a and X_b are depicted along the x-axis and y-axis, respectively. The aggregate AVG(Y) is depicted by the color intensity for each block (i.e., each X_a, X_b combination) of the heatmap. Give a continuous color scale, the higher the value of AVG(Y), the higher the color intensity. The complete set of queries (including WHERE clauses and other aggregates) that are supported by INCVISAGE can be found in Section 3.5—we focus on the simple setting for now.

Note that we are implicitly focusing on X_a and X_b that are *ordinal*, i.e., have an order associated with them so that it makes sense to visualize them as a trendline or a heatmap. As we will demonstrate subsequently, this order is crucial in letting us approximate portions of the visualization that share similar behavior.

Sampling Engine. We assume that we have a sampling engine that can efficiently retrieve random tuples from R corresponding to different values of the dimension attribute(s) X_a and/or X_b (along with optional predicates from a WHERE). Focusing on Q_T for now, given a certain value of $X_a = a_i$, our engine provides us a random tuple that satisfies $X_a = a_i$. Then, by looking up the value of Y corresponding to this tuple, we can get an estimate for the average of Y for $X_a = a_i$. Our sampling engine is drawn from Kim et al. [30] and maintains an in-memory bitmap on the dimension attributes, allowing us to quickly identify tuples matching arbitrary conditions [31]. Bitmaps are highly compressible and effective for read-only workloads [32, 49, 48], and have been applied recently to sampling for approximate generation of bar charts [30].

Thus, given our sampling engine, we can retrieve samples of Y given $X_a = a_i$ (or $X_a = a_i \wedge X_b = b_i$ for heat maps). We call the multiset of values of Y corresponding to $X_a = a_i$ across all tuples to be a *group*. This allows us to say that we are *sampling from a group*, where implicitly we mean we are sampling the corresponding tuples and retrieving the Y value.

Next, we describe our problem of incrementally generating visualizations more formally. We focus on trendlines (i.e., Q_T); the corresponding definitions and techniques for heatmaps are described in our extended technical report [2].

2.2 Incremental Visualizations

From this point forward, we describe the concepts in the context of our visualizations in row 2 of Figure 1.

Segments and k-Segment Approximations. We denote the cardinality of our group-by dimension attribute X_a as m, i.e., $|X_a| = m$. In Figure 1 this value is 36. At all time points over the course of visualization generation, we display one value of AVG(Y) corresponding to each group $x_i \in X_a, i \in 1 \ldots m$—thus, the user is always shown a complete trendline visualization. To approximate our trendlines, we use the notion of *segments* that encompass multiple groups. We define a segment as follows:

Definition 1. *A segment S corresponds to a pair (I, η), where η is a value, while I is an interval $I \subseteq [1, m]$ spanning a consecutive sequence of groups $x_i \in X_a$.*

For example, the segment S $([2, 4], 0.7)$ corresponds to the interval of x_i corresponding to x_2, x_3, x_4, and has a value of 0.7. Then, a k-segment approximation of a visualization comprises k disjoint segments that span the entire range of x_i, $i = 1 \ldots m$. Formally:

Definition 2. *A* k-segment *approximation is a tuple $L_k = \big(S_1, \ldots, S_k\big)$ such that the segments $S_1, \ldots, S_k$ partition the interval $[1, m]$ into k disjoint sub-intervals.*

In Figure 1, at t_2, we display a 2-segment approximation, with segments $([1, 30], 0.4)$ and $([31, 36], 0.8)$; and at t_7, we display a 7-segment approximation, comprising $([1, 3], 0.8)$, $([4, 14], 0.4)$, …, and $([35, 36], 0.7)$. When the number of segments is unspecified, we simply refer to this as a *segment* approximation.

Incrementally Improving Visualizations. We are now ready to describe our notion of incrementally improving visualizations.

Definition 3. *An* incrementally improving visualization *is defined to be a sequence of m segment approximations, $(L_1, \ldots, L_m)$, where the ith item $L_i, i > 1$ in the sequence is a i-segment approximation, formed by selecting one of the segments in the $(i-1)$-segment approximation L_{i-1}, and dividing that segment into two.*

Thus, the segment approximations that comprise an incrementally improving visualization have a very special relationship to each other: each one is a *refinement* of the previous, revealing one new *feature* of the visualization and is formed by dividing one of the segments S in the i-segment approximation into two new segments to give an $(i + 1)$-segment approximation: we call this process *splitting a segment*. The group within $S \in L_i$ immediately following which the split occurs is referred to as a *split group*. Any group in the interval $I \in S$ except for the last group can be chosen as a *split group*. As an example, in Figure 1, the entire second row corresponds to an incrementally improving visualization, where the 2-segment approximation is generated by taking the segment in the 1-segment approximation corresponding to $([1, 36], 0.5)$, and splitting it at group 30 to give $([1, 30], 0.4)$ and $([31, 36], 0.8)$. Therefore, the *split group* is 30. The reason why we enforce two neighboring segment approximations to be related in this way is to ensure that there is *continuity* in the way the visualization is generated, making it a smooth user experience. If, on the other hand, each subsequent segment approximation had no relationship to the previous one, it could be a very jarring experience for the user with the visualizations varying drastically, making it hard for them to be confident in their decision making.

Underlying Data Model and Output Model. To characterize the performance of an incrementally improving visualization, we need a model for the underlying data. We represent the underlying data as a sequence of m distributions $D_1, \ldots, D_m$ with means $\mu_1, \ldots, \mu_m$ where, $\mu_i =$ AVG(Y) for $x_i \in X_a$. To generate our incrementally improving visualization and its constituent segment approximations, we draw samples from distributions $D_1, \ldots, D_m$. Our goal is to approximate the mean values $(\mu_1, \ldots, \mu_m)$ while taking as few samples as possible.

The output of a k-segment approximation L_k can be represented alternately as a sequence of values $(\nu_1, \ldots, \nu_m)$ such that ν_i is equal to the value corresponding to the segment that encompasses x_i, i.e., $\forall_{x_i \in S_j} \nu_i = \eta_j$, where $S_j(I, \eta_j) \in L_k$. By comparing $(\nu_1, \ldots, \nu_m)$ with $(\mu_1, \ldots, \mu_m)$, we can evaluate the error corresponding to a k-segment approximation, as we describe later.

Incrementally Improving Visualization Generation Algorithm. Given our data model, an incrementally improving visualization generation algorithm proceeds in *iterations*: at the *i*th iteration, this algorithm takes some samples from the distributions $D_1, \ldots, D_m$, and then at the end of the iteration, it outputs the i-segment approximation L_i. We denote the number of samples taken in iteration i as N_i. When L_m is output, the algorithm terminates.

2.3 Characterizing Performance

There are multiple ways to characterize the performance of incrementally improving visualization generation algorithms.

Sample Complexity, Interactivity and Wall-Clock Time. The first and most straightforward way to evaluate performance is by measuring the samples taken in each iteration k, N_k, i.e., the *sample complexity*. Since the time taken to acquire the samples is proportional to the number of samples in our sampling engine (as shown in [30]), this is a proxy for the time taken in each iteration. Recent work has identified 500ms as a "rule of thumb" for interactivity in exploratory visual data analysis [36], beyond which analysts end up getting frustrated, and as a result explore fewer hypotheses. To enforce this rule of thumb, we can ensure that our algorithms take only as many samples per iteration as is feasible within 500ms — a time budget. We also introduce a new metric called interactivity that quantifies the overall user experience:

$$\lambda = \frac{\sum_{k=1}^{m} N_k \times (m - k + 1)}{k'}$$

where N_k is the number of samples taken at iteration k and k' is the number of iterations where $N_k > 0$. The larger the λ, the smaller the interactivity: this measure essentially captures the average waiting time across iterations where samples are taken. We explore the measure in detail in Section 3.4 and 5.5. A more direct way to evaluate performance is to measure the wall-clock time for each iteration.

Error Per Iteration. Since our incrementally improving visualization algorithms trade off taking fewer samples to return results faster, it can also end up returning segment approximations that are incorrect. We define the ℓ_2 *squared error* of a k-segment approximation L_k with output sequence $(\nu_1, \ldots, \nu_m)$ for the distributions

$D_1, \ldots, D_m$ with means $\mu_1, \ldots, \mu_m$ as

$$\mathrm{err}(L_k) = \frac{1}{m} \sum_{i=1}^{m} (\mu_i - \nu_i)^2 \tag{1}$$

We note that there are several reasons a given k-segment approximation may be erroneous with respect to the underlying mean values $(\mu_1, \ldots, \mu_m)$: (1) We are representing the data using k-segments as opposed to m-segments. (2) We use incremental refinement: each segment approximation builds on the previous, possibly erroneous estimates. (3) The estimates for each group and each segment may be biased due to sampling.

These types of error are all unavoidable — the first two reasons enable a visualization that incrementally improves over time, while the last one occurs whenever we perform sampling: (1) While a k-segment approximation does not capture the data exactly, it provides a good approximation preserving visual features, such as the overall major trends and the regions with a large value. Moreover, computing an accurate k-segment approximation requires fewer samples and therefore faster than an accurate m-segment approximation. (2) Incremental refinement allows users to have a fluid experience, without the visualization changing completely between neighboring approximations. At the same time, is not much worse in error than visualizations that change completely between approximations, as we will see later.

2.4 Problem Statement

The goal of our incrementally improving visualization generation algorithm is, at each iteration k, generate a k-segment approximation that is not just "close" to the best possible refinement at that point, but also takes the least number of samples to generate such an approximation. Further, since the decisions made by our algorithm can vary depending on the samples retrieved, we allow the user to specify a failure probability δ, which we expect to be close to zero, such that the algorithm satisfies the "closeness" guarantee with probability at least $1 - \delta$.

Problem 1. *Given a query Q_T, and the parameters δ, ϵ, design an incrementally improving visualization generation algorithm that, at each iteration k, returns a k-segment approximation while taking as few samples as possible, such that with probability $1 - \delta$, the error of the k-segment approximation L_k returned at iteration k does not exceed the error of the best k-segment approximation formed by splitting a segment of L_{k-1} by no more than ϵ.*

3. VISUALIZATION ALGORITHMS

In this section, we gradually build up our solution to Problem 1. We start with the ideal case when we know the means of all of the distributions up-front, and then work towards deriving an error guarantee for a single iteration. Then, we propose our incrementally improving visualization generation algorithm *ISplit* assuming the same guarantee across all iterations. We further discuss how we can tune the guarantee across iterations in Section 3.4. We consider extensions to other query classes in Section 3.5. We can derive similar algorithms and guarantees for heatmaps [2]. All the proofs can be found in the technical report [2].

3.1 Case 1: The Ideal Scenario

We first consider the ideal case where the means $\mu_1, \ldots, \mu_m$ of the distributions $D_1, \ldots, D_m$ are known. Then, our goal reduces to obtaining the best k segment approximation of the distributions at iteration k, while respecting the refinement restriction. Say the incrementally improving visualization generation algorithm has obtained a k-segment approximation L_k at the end of iteration k. Then, at iteration $(k+1)$, the task is to identify a segment $S_i \in L_k$ such that splitting S_i into two new segments T and U minimizes the error of the corresponding $(k+1)$-segment approximation L_{k+1}. We describe the approach, followed by its justification.

Approach. At each iteration, we split the segment $S_i \in L_k$ into T and U that maximizes the quantity $\frac{|T|\cdot|U|}{|S_i|\cdot m}(\mu_T - \mu_U)^2$. Intuitively, this quantity—defined below as the *improvement potential* of a refinement—picks segments that are large, and within them, splits where we get roughly equal sized T and U, with large differences between μ_T and μ_U.

Justification. The ℓ_2 *squared error* of a segment S_i (I_i, η_i), where $I_i = [p, q]$ and $1 \le p \le q \le m$, for the distributions $D_p, \ldots, D_q$ with means $\mu_p, \ldots, \mu_q$ is

$$\mathrm{err}(S_i) = \frac{1}{(q-p+1)} \sum_{j=p}^{q} (\mu_j - \eta_i)^2 = \frac{1}{|S_i|} \sum_{j \in S_i} (\mu_j - \eta_i)^2$$

Here, $|S_i|$ is the number of groups (distributions) encompassed by segment S_i. When the means of the distributions are known, $\mathrm{err}(S_i)$ will be minimized if we represent the value of segment S_i as the mean of the groups encompassed by S_i, i.e., setting $\eta_i = \mu_{S_i} = \sum_{j \in S_i} \mu_j / |S_i|$ minimizes $\mathrm{err}(S_i)$. Therefore, in the ideal scenario, the error of the segment S_i is

$$\mathrm{err}(S_i) = \frac{1}{|S_i|} \sum_{j \in S_i} (\mu_j - \eta_i)^2 = \frac{1}{|S_i|} \sum_{j \in S_i} \mu_j^2 - \mu_{S_i}^2 \tag{2}$$

Then, using Equation 1, we can express the ℓ_2 *squared error* of the k-segment approximation L_k as follows:

$$\mathrm{err}(L_k) = \frac{1}{m} \sum_{i=1}^{m} (\mu_i - \nu_i)^2 = \sum_{i=1}^{k} \frac{|S_i|}{m} \mathrm{err}(S_i)$$

Now, L_{k+1} is obtained by splitting a segment $S_i \in L_k$ into two segments T and U. Then, the error of L_{k+1} is:

$$\begin{aligned} \mathrm{err}(L_{k+1}) &= \mathrm{err}(L_k) - \frac{|S_i|}{m}\mathrm{err}(S_i) + \frac{|T|}{m}\mathrm{err}(T) + \frac{|U|}{m}\mathrm{err}(U) \\ &= \mathrm{err}(L_k) - \frac{|T| \cdot |U|}{|S_i| \cdot m}(\mu_T - \mu_U)^2. \end{aligned}$$

We use the above expression to define the notion of *improvement potential*. The *improvement potential* of a segment $S_i \in L_k$ is the minimization of the error of L_{k+1} obtained by splitting S_i into T and U. Thus, the *improvement potential* of segment S_i relative to T and U is

$$\Delta(S_i, T, U) = \frac{|T| \cdot |U|}{|S_i| \cdot m}(\mu_T - \mu_U)^2 \tag{3}$$

For any segment $S_i = (I_i, \eta_i)$, every group in the interval I_i except the last one can be chosen to be the *split group* (see Section 2.2). The *split group* maximizing the *improvement potential* of S_i, minimizes the error of L_{k+1}. The maximum *improvement potential* of a segment is expressed as follows:

$$\Delta^\star(S_i) = \max_{T,U \subseteq S_i} \Delta(S_i, T, U) = \max_{T,U \subseteq S_i} \frac{|T| \cdot |U|}{|S_i| \cdot m}(\mu_T - \mu_U)^2$$

Lastly, we denote the *improvement potential* of a given L_{k+1} by $\phi(L_{k+1}, S_i, T, U)$, where $\phi(L_{k+1}, S_i, T, U) = \Delta(S_i, T, U)$. Therefore, the maximum *improvement potential* of L_{k+1}, $\phi^\star(L_{k+1}) = \max_{S_i \subseteq L_k} \Delta^\star(S_i)$. When the means of the distributions are known, at iteration $(k+1)$, the optimal algorithm simply selects the refinement corresponding to $\phi^\star(L_{k+1})$, which is the segment approximation with the maximum improvement potential.

3.2 Case 2: The Online-Sampling Scenario

We now consider the case where the means $\mu_1, \ldots, \mu_m$ are *unknown*. Similar to the previous case, we want to identify a segment $S_i \in L_k$ such that splitting S_i into T and U results in the maximum *improvement potential*. We will first describe our approach for a given iteration assuming samples have been taken, then we will describe our approach for selecting samples.

3.2.1 Selecting the Refinement Given Samples

Approach. Unfortunately, since the means are unknown, we cannot measure the exact improvement potential, so we minimize the *empirical* improvement potential based on samples seen so far. Analogous to the previous section, we simply pick the refinement that maximizes $\frac{|T|\cdot|U|}{|S_i|\cdot m}(\widetilde{\mu}_T - \widetilde{\mu}_U)^2$, where the $\widetilde{\mu}$s are the empirical estimates of the means.

Justification. At iteration $(k+1)$, we first draw samples from the distributions $D_1, \ldots, D_m$ to obtain the estimated means $\tilde{\mu}_1, \ldots, \tilde{\mu}_m$. For each $S_i \in L_k$, we set its value to $\eta_i = \widetilde{\mu}_{S_i} = \sum_{j \in S_i} \tilde{\mu}_j / |S_i|$, which we call the *estimated mean* of S_i. For any refinement L_{k+1} of L_k, we then let the *estimated improvement potential* of L_{k+1} be

$$\widetilde{\phi}(L_{k+1}, S_i, T, U) = \frac{|T|}{m}\widetilde{\mu}_T^{\,2} + \frac{|U|}{m}\widetilde{\mu}_U^{\,2} - \frac{|S|}{m}\widetilde{\mu}_{S_i}^{\,2} = \frac{|T|\cdot|U|}{|S_i|\cdot m}(\widetilde{\mu}_T - \widetilde{\mu}_U)^2$$

For simplicity we denote $\phi(L_{k+1}, S_i, T, U)$ and $\widetilde{\phi}(L_{k+1}, S_i, T, U)$ as $\phi(L_{k+1})$ and $\widetilde{\phi}(L_{k+1})$, respectively.

Our goal is to get a guarantee for $\mathrm{err}(L_{k+1})$ is relative to $\mathrm{err}(L_k)$. Instead of a guarantee on the actual error err, for which we would need to know the means of the distributions, our guarantee is instead on a new quantity, err', which we define to be the *empirical* error. Given $S_i = (I_i, \eta_i)$ and Equation 2, err' is defined as follows: $\mathrm{err}'(S_i) = \frac{1}{|S_i|}\sum_{j \in S_i} \mu_j^2 - \eta_i^2$. Although $\mathrm{err}'(S_i)$ is not equal to $\mathrm{err}(S_i)$ when $\eta_i \neq \mu_S$, $\mathrm{err}'(S_i)$ converges to $\mathrm{err}(S_i)$ as η_i gets closer to μ_S (i.e., as more samples are taken). Similarly, the error of k-segment approximation $\mathrm{err}'(L_k)$ converges to $\mathrm{err}(L_k)$. We show experimentally (see Section 5) that optimizing for $\mathrm{err}'(L_{k+1})$ gives us a good solution of $\mathrm{err}(L_{k+1})$ itself.

To derive our guarantee on err', we need one more piece of terminology. At iteration $(k+1)$, we define $T\ (I, \eta)$, where $I = [p, q]$ and $1 \leq p \leq q \leq m$ to be a *boundary segment* if either p or q is a *split group* in L_k. In other words, at the iteration $(k+1)$, all the segments in L_k and all the segments that may appear in L_{k+1} after splitting a segment are called boundary segments. Next, we show that if we estimate the boundary segments accurately, then we can find a split which is very close to the best possible split.

Theorem 1. *If for every boundary segment T of the k-segment approximation L_k, we obtain an estimate $\widetilde{\mu}_T$ of the mean μ_T that satisfies $\left|\widetilde{\mu}_T^2 - \mu_T^2\right| \leq \frac{\epsilon m}{6|T|}$, then the refinement $L_{k+1}^{\dagger}$ of L_k that maximizes the estimated value $\widetilde{\phi}(L_{k+1}^{\dagger})$ will have error that exceeds the error of the best refinement L_{k+1}^{*} of L_k by at most* $\mathrm{err}'(L_{k+1}^{\dagger}) - \mathrm{err}'(L_{k+1}^{*}) \leq \epsilon$.

3.2.2 Determining the Sample Complexity

To achieve the guarantee for Theorem 1, we need to retrieve a certain number of samples from each of the distributions.

Approach. Perhaps somewhat surprisingly, we find that we need to draw a constant $C = \left\lceil \frac{288\, a\, \sigma^2}{\epsilon^2 m} \ln\left(\frac{4m}{\delta}\right) \right\rceil$ from each distribution D_i to satisfy the requirements of Theorem 1. (We will define these parameters subsequently.) Thus, our sampling approach is remarkably simple—and is essentially *uniform sampling*—plus, as we show in the next subsection, other approaches cannot provide significantly better guarantees. What is not simple, however, is showing that taking C samples allows us to satisfy the requirements for Theorem 1. Another benefit of uniform sampling is that we can switch between showing our segment approximations or showing the actual running estimates of each of the groups (as in online aggregation [19])—for the latter purpose, uniform sampling is trivially optimal.

Justification. To justify our choice, we assume that the data is generated from a *sub-Gaussian distribution* [42]. Sub-Gaussian distributions form a general class of distributions with a strong tail decay property, with its tails decaying at least as rapidly as the tails of a Gaussian distribution. This class encompasses Gaussian distributions as well as distributions where extreme outliers are rare—this is certainly true when the values derive from real observations that are bounded. We validate this in our experiments. Therefore, we represent the distributions $D_1, \ldots, D_m$ by m sub-Gaussian distributions with mean μ_i and sub-Gaussian parameter σ.

Given this generative assumption, we can determine the number of samples required to obtain an estimate with a desired accuracy using Hoeffding's inequality [21]. Given Hoeffding's inequality along with the union bound, we can derive an upper bound on the number of samples we need to estimate the mean μ_i of each x_i.

Lemma 1. *For a fixed $\delta > 0$ and a k-segment approximation of the distributions $D_1, \ldots, D_m$ represented by m independent random samples $x_1, \ldots, x_m$ with sub-Gaussian parameter σ^2 and mean $\mu_i \in [0, a]$ if we draw $C = \left\lceil \frac{288\, a\, \sigma^2}{\epsilon^2 m} \ln\left(\frac{4m}{\delta}\right) \right\rceil$ samples uniformly from each x_i, then with probability at least $1 - \delta$, $\left|\widetilde{\mu}_T^2 - \mu_T^2\right| \leq \frac{\epsilon m}{6|T|}$ for every boundary segment T of L_k.*

3.2.3 Deriving a Lower bound

We can derive a lower bound for the sample complexity of any algorithm for Problem 1:

Theorem 2. *Say we have a dataset D of m groups with means $(\mu_1, \mu_2, \ldots, \mu_m)$. Assume there exists an algorithm $\mathcal{A}$ that finds a k-segment approximation which is ϵ-close to the optimal k-segment approximation with probability 2/3. For sufficiently small ϵ, $\mathcal{A}$ draws $\Omega(\sqrt{k}/\epsilon^2)$ samples.*

The theorem states that any algorithm that outputs a k-segment approximation of which is ϵ-close to a dataset has to draw $O(\sqrt{k}/\epsilon^2)$ samples from the dataset.

3.3 The *ISplit* Algorithm

We now present our incrementally improving visualization generation algorithm *ISplit*. Given the parameters ϵ and δ, *ISplit* maintains the same guarantee of error (ϵ) in generating the segment approximations in each iteration. Theorem 1 and Lemma 1 suffice to show that the *ISplit* algorithm is a greedy approximator, that, at each iteration identifies a segment approximation that is at least ϵ close to the best segment approximation for that iteration.

```
Data: $X_a, Y, \delta, \epsilon$
Start with the 1-segment approximator $L = (L_1)$.
for $k = 2, \ldots, m$ do
    $L_{k-1} = (S_1, \ldots, S_{k-1})$.
    for each $S_i \in L_{k-1}$ do
        for each group $q \in S_p$ do
            Draw $C$ samples uniformly. Compute mean $\tilde{\mu}_q$
        end
        Compute $\widetilde{\mu}_{S_i} = \frac{1}{|S_i|}\sum_{q \in S_i} \tilde{\mu}_q$
    end
    Find $(T, U) = \mathrm{argmax}_{i;\, T,U \subseteq S_i} \frac{|T|\cdot|U|}{|S_i|\cdot m}(\widetilde{\mu}_T - \widetilde{\mu}_U)^2$.
    Update $L_{k+1} = S_1, \ldots, S_{i-1}, T, U, S_{i+1}, \ldots, S_k$.
end
```

Algorithm 1: *ISplit*

The parameter ϵ is often difficult for end-users to specify. Therefore, in INCVISAGE, we allow users to instead explicitly specify an expected *time budget* per iteration, B—as explained in Section 2.3, the number of samples taken determines the time taken per iteration, so we can reverse engineer the number of samples from B. So, given the sampling rate f of the sampling engine (see Section 2.1) and B, the sample complexity per iteration per group is simply $C = B \times f/m$. Using Lemma 1, we can compute the corresponding ϵ. Another way of setting B is to use standard rules of thumb for interactivity (see Section 2.3).

3.4 Tuning ϵ Across Iterations

So far, we have considered only the case where the algorithm takes the same number of samples C per group across iterations and does not reuse the samples across different iterations. If we instead reuse the samples drawn in previous iterations, the error at iteration k is $\epsilon_k^2 = \frac{288\,a\,\sigma^2}{mkC} \ln\left(\frac{4m}{\delta}\right)$, where $k\,C$ is the total number of samples drawn so far. Therefore, the decrease in the error up to iteration k, ϵ_k, from the error up to iteration $(k-1)$, ϵ_{k-1}, is $\sqrt{(k-1)/k}$, where $\epsilon_k = \sqrt{(k-1)/k}\,\epsilon_{k-1}$. This has the following effect: later iterations both take the same amount of time as previous ones, and produce only minor updates of the visualization.

Sampling Approaches. This observation indicates that we can obtain higher interactivity with only a small effect on the accuracy of the approximations by considering variants of the algorithm that decrease the number of samples across iterations instead of drawing the same number of samples in each iteration. We consider three natural approaches for determining the number of samples we draw at each iteration: linear decrease (i.e., reduce the number of samples by β at each iteration), geometric decrease (i.e., divide the number of samples by α at each iteration), and all-upfront (i.e., non-interactive) sampling. To compare these different approaches to the constant (uniform) sampling approach and amongst each other, we first compute the total sample complexity, interactivity, and error guarantees of each of them as a function of their other parameters. Letting T_k denote the total number of samples drawn in the first k iterations, the *interactivity* of a sampling approach defined in Section 2.3 can be written as: $\lambda = \frac{\sum_{k=1}^{m} T_k}{k'}$, where k' is the number of iterations we take non-zero samples. The error guarantees we consider are, as above, the average error over all iterations. This error guarantee is

$$\text{err} = \sum_{i=1}^{m} \frac{\epsilon_k^2}{m} = \sum_{i=1}^{m} \frac{288\,a\,\sigma^2}{m^2 T_k} \ln\left(\frac{4m}{\delta}\right) = \frac{288\,a\,\sigma^2}{m^2} \ln\left(\frac{4m}{\delta}\right) \sum_{i=1}^{m} \frac{1}{T_k}$$

We provide evidence that the estimated err and λ are similar to err and λ on real datasets in Section 5.5.2. We are now ready to derive the expressions for both error and interactivity for the sampling approaches mentioned earlier.

Expressions for Error and Interactivity. The analytical expressions of both interactivity and error for all of these four approaches are shown in Table 1 and derived in the technical report [2]. In particular, for succinctness, we have left the formulae for Error for geometric and linear decrease as is without simplification; on substituting T_k into the expression for Error, we obtain fairly complicated formulae with dozens of terms, including multiple uses of the q-digamma function Ψ [25]—for the geometric decrease case, the Euler-Mascheroni constant [44]—for the linear decrease case, and the Harmonic number H_m—for the constant sampling case: these expressions are provided in the technical report [2].

Comparing the Sampling Approaches. We first examine the all-upfront sampling approach. We find that the all-upfront sampling approach is strictly worse than constant sampling (which is a special case of linear and geometric decrease).

Theorem 3. *If for a setting of parameters, a constant sampling approach and an all-upfront sampling approach have the same interactivity, then the error of constant sampling is strictly less than the error of all-upfront sampling.*

Our experimental results in Section 5.5 suggests that the geometric decrease approach with the optimal choice of parameter α^* has better error than not only the all-upfront approach but the linear decrease and constant sampling approaches as well. This remains an open question, but when we fix the initial sample complexity N_1 (which is proportional to the bound on the maximum time taken per iteration as provided by the user), we can show that geometric decrease with the right choice of parameter α does have the optimal interactivity among the three interactive approaches.

Theorem 4. *Given N_1, the interactivity of geometric decrease with parameter $\alpha^* = (N_1 - 1)^{1/(m-1)}$ is strictly better than the interactivity of any linear decrease approach and constant sampling.*

The proof is established by first showing that for both linear and geometric decrease, the optimal interactivity is obtained for an explicit choice of decrease parameter that depends only on the initial number of samples and the total number of iterations.

Lemma 2. *In geometric sampling with a fixed N_1, $\alpha^* = (N_1 - 1)^{1/(m-1)}$ has the optimal interactivity and it has smaller error than all of the geometric sampling approaches with $\alpha > \alpha^*$.*

Lemma 3. *In linear sampling with a fixed N_1, $\beta^* = (N_1 - 1)/(m-1)$ has the optimal interactivity and it has strictly smaller error than all of the linear sampling approaches with $\beta > \beta^*$.*

Optimal α and Knee Region. As shown in Lemma 2, given N_1, we can compute the optimal decrease parameter α^*, such that any value of $\alpha > \alpha^*$ results in higher error and lesser interactivity (higher λ). This behavior results into the emergence of a knee region in the error-interactivity curve which is confirmed in our experimental results (see Section 5.5). Essentially, starting from $\alpha = 1$ any value $\alpha \leq \alpha^*$ has smaller error than any value $\alpha > \alpha^*$. Therefore, for any given N_1 there is an optimal region $[1, \alpha^*]$. For example, for $N_1 = 5000, 25000, 5000$, and 10000, the optimal range of α is $[1, 1.024]$, $[1, 1.028]$, $[1, 1.03]$, and $[1, 1.032]$, respectively. By varying α along the optimal region one can either optimize for error or interactivity. For example, starting with $\alpha = 1$ as $\alpha \to \alpha^*$ we trade accuracy for interactivity.

3.5 Extensions

The incrementally improving visualization generation algorithm described previously for simple queries with the AVG aggregation function can also be extended to support aggregations such as COUNT and SUM, as well as additional predicates (via a WHERE clause)—details and proofs can be found in the technical report [2].

3.5.1 The COUNT *Aggregate Function*

Given that we know the total number of tuples in the database, estimating the COUNT aggregate function essentially corresponds to the problem of estimating the fraction of tuples τ_i that belong to each group i. We focus on the setting below when we only use our bitmap index. We note that approximation techniques for COUNT queries have also been studied previously [8, 24, 26], for the case when no indexes are available.

Using our sampling engine, we can estimate the fractions τ_i by scanning the bitmap index for each group. When we retrieve another sample from group i, we can also estimate the number of tuples we needed to skip over to reach the tuple that belongs to group i—the indices allow us to estimate this information directly, providing us an estimate for τ_i, i.e., $\tilde{\tau}_i$.

Theorem 5. *With an expected total number of samples $C_{count} = m + \lceil \gamma^{-2} \ln(2m/\delta)/2 \rceil$, the $\tilde{\tau}_i, \forall i$ can be estimated to within a factor γ, i.e., $|\tilde{\tau}_i - \tau_i| \leq \gamma$ holds with probability at least $1 - \delta$.*

Essentially, the algorithm takes as many samples from each group until the index of the sample is $\geq \lceil \gamma^{-2} \ln(2m/\delta)/2 \rceil$. We can show that overall, the expected number of samples is bounded above by C_{count}. Since this number is small, we don't even need to incrementally estimate or visualize COUNT.

3.5.2 The SUM Aggregate Function

. There are two settings we consider for the SUM aggregate function: when the group sizes (i.e., the number of tuples in each group) are known, and when they are unknown.

Known Group Sizes. A simple variant of Algorithm 1 can also be used to approximate SUM in this case. Let n_i be the size of

Approach	decrease parameter	T_k	Error	Interactivity
Linear Decrease	β	$N_1 k - \frac{k\,(k-1)\,\beta}{2}$	$A \sum_{k=1}^{m} \frac{1}{T_k}$	$N_1 \left(m - \frac{k'-1}{2}\frac{N_1}{m}\right) + \frac{\beta}{6}\left[2k'^2 - 3k' + 1 - 3mk' + 3m\right]$
Geometric Decrease	α	$N_1 \frac{\alpha^k - 1}{\alpha^{k-1}(\alpha-1)}$	$A \sum_{k=1}^{m} \frac{1}{T_k}$	$\frac{N_1}{m}\left[\frac{m\,(\alpha-1)\,\alpha^m + \alpha^m - 1}{(\alpha-1)^2\,\alpha^{m-1}}\right]$
Constant Sampling	$\alpha = 1$, or $\beta = 0$	kN_1	$\frac{A \cdot H_m}{N_1}$	$\frac{N_1\,(m+1)}{2}$
All-upfront	—	$T_{k=1} = N_1$ and $T_{k>1} = 0$	$\frac{A \cdot m}{N_1}$	mN_1

Table 1: Expressions for error and interactivity for different sampling approaches. Here, $A = \frac{288\,a\,\sigma^2}{m^2} \ln\left(\frac{4m}{\delta}\right)$

group i and $\kappa = \max_j n_j$. As in the original setting, the algorithm computes the estimate $\tilde{\mu}_i$ of the average of the group. Then $\tilde{s}_i = n_i\,\tilde{\mu}_i$ is an estimate on the sum of each group i, namely s_i, that is used in place of the average in the rest of the algorithm. We have:

Theorem 6. *Assume we have $C_i = \lceil \frac{288a^2\sigma^2 m n_i^2}{\epsilon^2\kappa^2} \ln \frac{4m}{\delta} \rceil$ samples from group i. Then, the refinement $L^{\dagger}_{k+1}$ of L_k that maximizes the estimated improvement potential $\widetilde{\phi}^{+}(L^{\dagger}_{k+1})$ will have error that exceeds the error of the best refinement L^{*}_{k+1} of L_k by at most* $\mathrm{err}'^{+}(L^{\dagger}_{k+1}) - \mathrm{err}'^{+}(L^{*}_{k+1}) \leq \epsilon\kappa^2$.

Note that here we have $\epsilon\kappa^2$ instead of ϵ: while at first glance this may seem like an amplification of the error, it is not so: first, the scales are different—and visualizing the SUM is like visualizing AVG multiplied by κ—an error by one "pixel" for the former would equal κ times the error by one "pixel" for the latter but are visually equivalent; second, our error function uses a squared ℓ_2-like distance, explaining the κ^2.

Unknown Group Sizes. For this case, we simply employ the known group size case, once the COUNT estimation is used to first approximate the τ_i with $\gamma = \epsilon/24a$. We have $\tilde{s}_i = \tilde{\tau}_i \tilde{\mu}_i \kappa_t$, where κ_t denotes the total number of items: $\sum_{i=1}^{m} n_i$.

Theorem 7. *Assume we have $C_i = \lceil \frac{1152\,a^2\sigma^2\tilde{\tau}_i^2}{\epsilon^2 m} \ln \frac{4m}{\delta} \rceil$ samples from group i. Then, the refinement $L^{\dagger}_{k+1}$ of L_k that maximizes the estimated $\widetilde{\phi}^{+}(L^{\dagger}_{k+1})$ will have error that exceeds the error of the best refinement L^{*}_{k+1} of L_k by at most* $\mathrm{err}' + (L^{\dagger}_{k+1}) - \mathrm{err}' + (L^{*}_{k+1}) \leq \epsilon\kappa_t^2$.

$$\mathtt{Q_T} \;=\; \mathtt{SELECT\ X_a, AVG(Y)\ FROM\ R\ GROUP}$$
$$\mathtt{BY\ X_a\ ORDER\ BY\ X_a\ WHERE\ Pred}$$

4. INCVISAGE SYSTEM ARCHITECTURE

The INCVISAGE client is a web-based front-end that captures user input and renders visualizations produced by the INCVISAGE back-end. The INCVISAGE back-end is composed of three components: (A) a *query parser*, which is responsible for parsing the input query Q_T or Q_H (see Section 2.1), (B) a *view processor*, which executes *ISplit* (see Section 3), and (C) a *sampling engine* which returns the samples requested by the *view processor* at each iteration. As discussed in Section 2.1, INCVISAGE uses a bitmap-based sampling engine to retrieve a random sample of records matching a set of ad-hoc conditions [30]. At the end of each iteration, the *view processor* sends the visualizations generated to the front-end encoded in *json*. To reduce the amount of json data transferred, only the changes in the visualization are communicated to the front-end.

The front-end is responsible for capturing user input and rendering visualizations generated by the INCVISAGE back-end. The visualizations (i.e., the segment approximations) generated by the back-end incrementally improve over time, but the users can pause and replay the visualizations on demand. The details of the front-end and screenshots, along with an architecture diagram can be found the technical report [2]

5. PERFORMANCE EVALUATION

We evaluate our algorithms on three primary metrics: the error of the visualizations, the degree of correlation with the "best" algorithm in choosing the split groups, and the runtime performance. We also discuss how the choice of the initial samples N_1 and sampling factor α impacts the error and interactivity.

5.1 Experimental Setup

Algorithms Tested. Each of the incrementally improving visualization generation algorithms that we evaluate performs uniform sampling, and take either B (time budget) and f (sampling rate of the sampling engine) as input, or ϵ (desired error threshold) and δ (the probability of correctness) as input, and in either case computes the required N_1 and α. The algorithms are as follows:

ISplit: At each iteration k, the algorithm draws $\frac{N_k}{m}$ samples uniformly from each group, where N_k is the total number of samples requested at iteration k and m is the total number of groups. *ISplit* uses the concept of *improvement potential* (see Section 3) to split an existing segment into two new segments.

RSplit: At each iteration k, the algorithm takes the same samples as *ISplit* but then selects a segment, and a split group within that, all at random to split.

ISplit-Best: The algorithm simulates the ideal case where the mean of all the groups are known upfront (see Section 3.1). The visualizations generated have the lowest possible error at any given iteration (i.e. for any k-segment approximation) among approaches that perform refinement of previous approximation.

DPSplit: We describe the details of this algorithm in our technical report [2], but at a high level, at each iteration k, this algorithm takes the same samples as *ISplit*, but instead of performing refinement, *DPSplit* recomputes the best possible k-segment approximation using dynamic programming. We include this algorithm to measure the impact of the iterative refinement constraint.

DPSplit-Best: This algorithm simulates the case where the means of all the groups are known upfront, and the same approach for producing k-segment approximations used by *DPSplit* is used. The visualizations generated have the lowest possible error among the algorithms mentioned above since they have advance knowledge of the means, and do not need to obey iterative refinement.

Name	Description	#Rows	Size (GB)	E	U
Sensor	Intel Sensor dataset [1].	2.2M	0.73		✓
FL	US Flight dataset [6] .	74M	7.2	✓	✓
T11	2011 NYC taxi trip data [4]	170M	6.3	✓	
T12	2012 NYC taxi trip data [4]	166M	4.5	✓	
T13	2013 NYC taxi trip data [4]	166M	4.3	✓	
WG	Weather data of US [7]	415M	27	✓	

Table 2: Datasets Used for the Experiments and User Studies. E = Experiments and U = User Studies (Section 6 and 7).

Datasets and Queries. The datasets used in the performance evaluation experiments and the user studies (Section 6 and Section 7) are shown in Table 2 and are marked by ticks (✓) to indicate where they were used. For the US flight dataset (FL) we show the results for the attribute *Arrival Delay*. Since all of the three years exhibit similar results for the NYC taxi dataset, we only present the results for the year 2011 (T11) for the attribute *Trip Time*. For the weather dataset, we show results for the attribute *Temperature*. The results for the other datasets are included in our technical report [2]. To verify our sub-Gaussian assumption, we generated a Q-Q plot [46] for each of the selected attributes to compare the distributions with Gaussian distributions. The FL, T11, T12, and T13 datasets all exhibit right-skewed Gaussian distributions while WG exhibits a truncated Gaussian distribution [2]. Unless stated explicitly, we use the same query in all the experiments—*calculate the average of an attribute at each day of the year*.

Metrics. We use the following metrics to evaluate the algorithms: Error: We measure the error of the visualizations generated at each iteration k via $\mathrm{err}(L_k)$ (see Section 2.3). The average error across iterations is computed as: $\widetilde{\mathrm{err}}(L_k) = \frac{1}{m}\sum_{k=1}^{m} \mathrm{err}(L_k)$.

Time: We also evaluate the *wall-clock execution time*.

Correlation: We use Spearman's ranked correlation coefficient to measure the degree of correlation between two algorithms in choosing the order of groups as split groups.

Interactivity: We use a new metric called interactivity (defined in Section 2.3) to select appropriate parameters for *ISplit*. Interactivity is essentially the average waiting time for generating the segment approximations. We explore the measure in Section 5.5.

Implementation Setup. We evaluate the runtime performance of all our algorithms on a bitmap-based sampling engine [31]. In addition, we implement a *SCAN* algorithm which performs a sequential scan of the entire dataset. This is the approach a traditional database system would use. Since both *ISplit* and *SCAN* are implemented on the same engine, we can make direct comparisons of execution times between the two algorithms. All our experiments are single threaded and are conducted on a HP-Z230-SFF workstation with an Intel Xeon E3-1240 CPU and 16GB memory running Ubuntu 12.04 LTS. We set the disk read block size to 256KB. Unless explicitly stated we assume the time budget $B = 500ms$ and use the parameter values of $N_1 = 25000$, $\alpha = 1.02$ (with a geometric decrease), and $\delta = 0.05$. All experiments were averaged over 30 trials.

5.2 Comparing Execution Time

In this section, we compare the *Wall Clock* time of *ISplit*, *DPSplit* and *SCAN* for the datasets mentioned in Table 2.

Summary: *ISplit* is several orders of magnitude faster than *SCAN* in revealing incremental visualizations. The completion time of *DPSplit* exceeds the completion time of even *SCAN*. Moreover, when generating the early segment approximations, *DPSplit* always exhibits higher latency compared to *ISplit*.

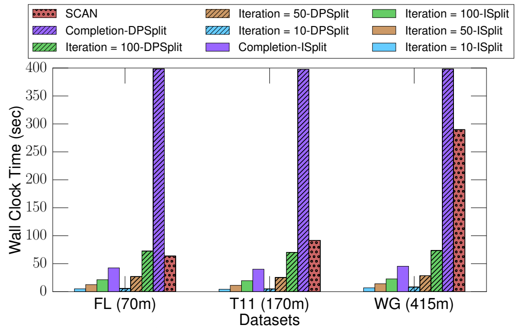


Figure 2: Comparison of Wall Clock Time.

We depict the wall-clock time in Figure 2 for three different datasets for *ISplit* and *DPSplit* at iteration 10, 50, and 100, and at completion, and for *SCAN*. First note that as the size of the dataset increases, the time for *SCAN* drastically increases since the amount of data that needs to be scanned increases. On the other hand, the time for completion for *ISplit* stays stable, and much smaller than *SCAN* for all datasets: on the largest dataset, the completion time is almost $\frac{1}{6}^{th}$ that of *SCAN*. When considering earlier iterations, *ISplit* performs even better, revealing the first 10, 50, and 100 segment approximations within ≈ 5 seconds, ≈ 13 seconds, and ≈ 22 seconds, respectively, for all datasets, allowing users to get insights early by taking a small number of samples. Compared to *SCAN*, the speed-up in revealing the first 10 features of the visualization is $\approx 12X$, $\approx 22X$, and $\approx 46X$ for the FL, T11 and WG datasets.

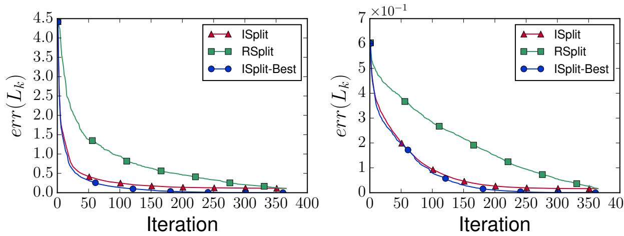
When comparing *ISplit* to *DPSplit*, we first note that *DPSplit* is taking the same samples as *ISplit*, so its differences in execution time are primarily due to computation time. We see some strange behavior in that while *DPSplit* is worse than *ISplit* for the early iterations, for completion it is much worse, and in fact worse than *SCAN*. The dynamic programming computation complexity depends on the number of segments. Therefore, the computation starts occupying a larger fraction of the overall wall-clock time for the latter iterations, rendering *DPSplit* impractical for latter iterations. Even for earlier iterations, *DPSplit* is worse than *ISplit*, revealing the first 10, 50, and 100 features within $\approx$ 7 seconds, $\approx$ 27 seconds, and $\approx$ 75 seconds, respectively, as opposed to 5, 13, and 22 for *ISplit*. As we will see later, this additional time does not come with a commensurate decrease in error, making *DPSplit* much less desirable than *ISplit* as an incrementally improving visualization generation algorithm.

5.3 Incremental Improvement Evaluation

We now compare the error for *ISplit* with *RSplit* and *ISplit-Best*.

Summary: *(a)* The error of *ISplit*, *RSplit*, and *ISplit-Best* reduce across iterations. At each iteration, *ISplit* exhibits lower error in generating visualizations than *RSplit*. *(b) ISplit* exhibits higher correlation with *ISplit-Best* in the choice of split groups, with ≥ 0.9 for any N_1 greater than 25000. *RSplit* has near-zero correlation overall.

Figure 3 depicts the iterations on the x-axis and the ℓ_2-error on the y axis of the corresponding segment approximations for each of the algorithms for two datasets (others are similar). For example, in Figure 3, at the first iteration, all three algorithms obtain the 1-segment approximation with roughly similar error; and as the number of iterations increase, the error continues to decrease. The error for *ISplit* is lower than *RSplit* throughout, for all datasets, justifying our choice of *improvement potential* as a good metric to guide splitting criteria. *ISplit-Best* has lower error than the other two, this is because *ISplit-Best* has access to the means for each group beforehand. We also explore the correlation of the split group order of *ISplit* and *RSplit* with *ISplit-Best* on varying the sample complexity. Due to space limitations, the results can be found in our technical report [2] where we show that *ISplit*'s split groups actually match those from *ISplit-Best*, even if the error is a bit higher.



a. FL

b. T11

Figure 3: ℓ_2 squared error of *ISplit*, *Rsplit*, and *ISplit-Best*.

5.4 Releasing the Refinement Restriction

We now compare *ISplit* with *DPSplit* and *DPSplit-Best*—we aim to evaluate the impact of the refinement restriction, and whether it leads to substantially lower error. Due to space limitations we again present the experiments in our technical report [2].

Summary: Given the same set of parameters, *DPSplit* and *ISplit* have roughly similar error; as expected *DPSplit-Best* has much lower error than both *ISplit* and *DPSplit*. Thus, in order to reduce the drastic variation of the segment approximations, while not having a significant impact on error, *ISplit* is a better choice than *DPSplit*.

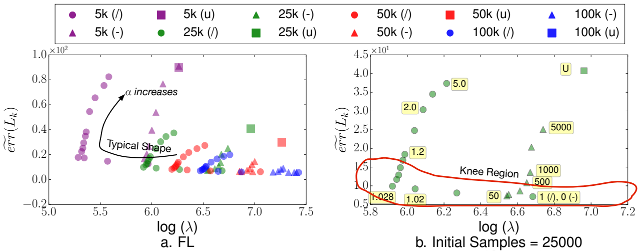
5.5 Optimizing for Error and Interactivity

So far, we have kept the sampling parameters fixed across algorithms; here we study the impact of these parameters. Specifically, we evaluate the impact of the initial sample (N_1) and sampling factor (α) on the error-interactivity trade-off.

Summary: We find: *(a)* Geometrically decreasing the sample complexity per iteration leads to higher interactivity. *(b)* Given the time budget (B), only a small set of sampling factors (α) improves interactivity. *(c)* The theoretical and experimental error-interactivity trade-off curves behave essentially the same, producing similarly shaped knee regions.

5.5.1 Parameter Selection

We now empirically evaluate the trade-off between error and interactivity. We focus on "decreasing" sampling factors, i.e., those that result in the sample complexity decreasing across iterations—we have found that "increasing" sampling factors lead to significantly worse λ (i.e., interactivity), while error is not reduced by much. We consider both geometric and linear decrease, as well as upfront sampling (Section 3.4). Figure 4 captures the trade-off between average error (across iterations) on the y axis and logarithm of interactivity, i.e., $\log \lambda$ on the x axis for the FL dataset (other datasets are similar)—focus on the circles and triangles for now. Each line in the chart corresponds a certain number of initial samples N_1, and either geometric decrease (denoted with a "/"), or a linear one (denoted with a "-"). Each point in each line corresponds to a specific value of α. For all lines, we find a similar behavior or shape as α increases, which we explain for $N_1 = 25000$ for geometric and linear decrease, depicted again in Figure 4d with α annotated. If we follow the geometric decrease points (circles) Figure 4b, we start at $I \approx 6.75$ at the point corresponding to $\alpha = 1$ for geometric decrease and 0 for linear decrease, and then as α is increased the points move to the left—indicating that the interactivity improves, while error increases slightly. Then, we have a *knee* in the curve—for any $\alpha > 1.028$, the trails start moving back to the right (lower interactivity) but also see a simultaneous increase in error. A similar knee is seen if we trace the linear decrease points (triangles)—note that $\alpha = 1$ is shared between the linear and geometric decrease—here, the sampling is constant across iterations. For other values of α depicted for the linear decrease points, this indicates the reduction in the number of samples per round, e.g., 50, 500, 1000. This behavior of a knee in the curve is seen for all N_1 values. We also find that for the same N_1, the linear decrease has worse interactivity compared to geometric decrease. Finally, on examining the upfront sampling scheme (squares), we find that both geometric decrease and linear decrease have much better interactivity and lower error.



a. FL

b. Initial Samples = 25000

Figure 4: Error-interactivity trade-off curve. (/) = Geometric decrease, (-) = Linear decrease, (u) = Upfront sampling

Overall, when selecting parameters, we would like to identify parameters that result in the favorable knee region of the curve. We find that $\alpha \in [1.0, 1.028]$, with N_1 relatively small helps us stay in this region empirically. We select $\alpha = 1.02$ to balance between error and interactivity if we set the maximum allowable delay per iteration $B = 500ms$ [36]. Based on our sampling rate, fetching 25000 samples takes around 500ms; thus we set $N_1 = 25000$. From our theoretical results in Section 3.4, the range of α for this N_1 was $[1, 1.028]$, so our experiments agree with the theory.

5.5.2 Simulations vs. Empirical Observations

We now simulate the error-interactivity trade-off curve for the sampling approaches using the expressions of error and interactivity obtained in Section 3.4. Figure 5 captures the trade-off between the upper bound of the error averaged (across iterations) on the y axis and logarithm of interactivity, i.e., $\log \lambda$ on the x axis for the FL and T11 dataset(other datasets are similar).

Each line in the chart corresponds a certain number of initial samples N_1. Each point in each line corresponds to a specific value of α. For all lines, we find a similar behavior as our empirical results—a knee shape emerges as α increases starting from 1. The theoretical value for the optimal decrease factor α^* is annotated for each initial sample N_1. Thus, the simulated trade-off curves generated based on the theory can mimic our empirical results. Simulations for upfront sampling and linear decrease similarly mimic empirical results and can be found in [2]. .

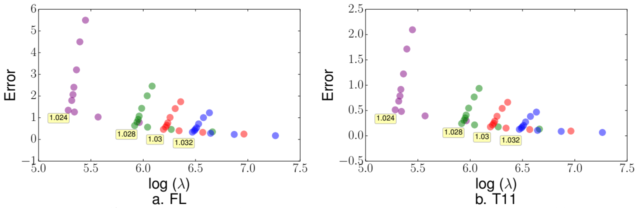
6. EVALUATION: INTERPRETABILITY

We now present an evaluation of the usability of IncVisage, and more broadly evaluate how users interpret and use incrementally improving visualizations.

6.1 Study Design and Participants

The study consisted of five phases: (a) an introduction phase that explained the essential components of IncVisage, (b) an exploration phase that allowed the participants to familiarize themselves with the interface by exploring a sensor dataset (see Section 5.1), (c) a quiz phase where the participants used the same interface to answer targeted questions on the flight dataset, (d) an interview to collect qualitative data during the quiz phase, and (e) a closing survey to obtain feedback on IncVisage. Our study was conducted prior to the development of *ISplit*, and was meant to assess the utility of incremental visualizations—nevertheless, we ensured that the interactivity criteria of 500ms per iteration was met [36].

We recruited 20 participants. Our participants included 11 graduate students (8 in computer science), one business undergraduate student, two researchers, and six university employees. All of the participants had experience with data analysis.



a. FL

b. T11

Figure 5: Simulation of the error-interactivity trade off curve.

6.2 The Quiz

We now explain the design of and findings from the quiz phase.

6.2.1 The Quiz Phase Design

The purpose of the quiz phase was to evaluate whether participants were willing to compromise accuracy to make rapid decisions when posed various types of questions.

We used two types of quiz questions: extrema-based (E1-7), and range-based (R1-7). These questions are listed in our technical report [2]. The extrema-based questions asked a participant to find the highest or lowest values in a visualization. The range-based questions asked a participant to estimate the average value over a time range (e.g., months, days of the week). The purpose of such a categorization is to evaluate the accuracy and confidence of participants in finding both specific points of interest (extrema) and patterns over a range (range) when given IncVisage. The extrema based questions were free form questions; the range based questions were multiple choice questions. To prevent order effects, ten participants started the quiz with heatmaps; the other ten started with trendlines. Additionally, we randomized the order of the questions for each participant. Next, we define the scoring function used in assessing quiz performances.

The Scoring Function. The scoring function relied on two variables: the iteration number at which the participant submitted an answer, and whether or not that answer was accurate. The participants were informed prior to the quiz phase that the score was based on these two variables, but the exact function was not provided. The score was computed as a product $S = P \cdot A$, where P was based on the iteration, and A on the accuracy. If a participant submitted an answer at iteration k, we set $P = \frac{m-k}{m}$, i.e., the fraction of the remaining number of iterations over the total number of iterations, m. To compute A, let c be the correct answer to a question, and let u be the answer submitted by the participant. The accuracy A of a multiple choice question is 0 if $u = c$ and 1 otherwise. The accuracy A of a free-form question is $1 - \frac{|u-c|}{|c|}$.

Interface Issues. Analyzing the quiz results proved more difficult than expected. Out of 280 total submissions, 10 submissions had interface issues—4 of those occurred due to ambiguity in the questions (*R1, R4, R7* in [2]), while others were due to mistakes made by the participants in selecting the visualization to be generated. The ambiguity arose from attribute names in the original dataset—to maintain consistency, instead of renaming these attributes on-the-fly to fix the ambiguity, we did not make any changes. We explicitly separate out interface issues when analyzing the results.

6.2.2 *Quantitative Analysis*

In discussing the participants' quiz performance, we first investigate their answer submission trends.

Summary: The majority of the submissions for both trendlines (75%) and heatmaps (77.5%) were within the first 25% of the total number of iterations. Even though the participants chose to trade accuracy for time in both cases, they did so with reasonably high accuracy.

Trading accuracy for time. Figure 6 shows a dot graph analysis of the participants' submission trends as a function of iteration. For both the trendline and heatmap visualizations, we separated the statistics for the correct and incorrect submissions (Figure 6a and 6b). Correct submissions are represented by *green* circles. Incorrect submissions are either represented by *blue* circles (interface issue) or *red* circles.

For trendlines, the majority of the submissions (75%) were made at around 25% of the total iterations, except for question *E4* (Figure 6a). Question *E4* asks for a *day of the year* that shows the highest *departure delay* across all years. During the study, we discovered that the departure delays for December 22 and December 23 were very similar. Due to the proximity of the values, these dates were not split into separate groups until a later iteration. Figure 6b shows the trends for heatmaps. Similar to trendlines, the majority of the participants (77.5%) chose to submit answers earlier (within 25% of the iterations) except for questions *R5* and *R7*, where once again the relevant heatmap block was split in a later iteration.

In our techreport, we also plot and analyze a scatterplot of accuracy and iteration. The submission trends indicate that participants opted for a higher reward and submitted their answers as early as possible, when the visualizations became stable or when obvious extrema emerged, trading accuracy for time.

Submission trends. For trendlines, the range-based questions were submitted earlier (75% of the submissions at % iteration ≈ 15%) compared to the extrema-based questions (75% of the submissions at %iteration ≈ 28%). This difference in submission trends across types may be due to the fact that the range based questions asked participants to find more coarse grained information (e.g., the delay statistics in a specific range) which may be easier than finding peaks and valleys. A box and whisker plot with these statistics can be found in [2]. We see the opposite trend for heatmaps; the extrema-based questions were submitted earlier compared to the range-based questions. We investigate this again in the next study.

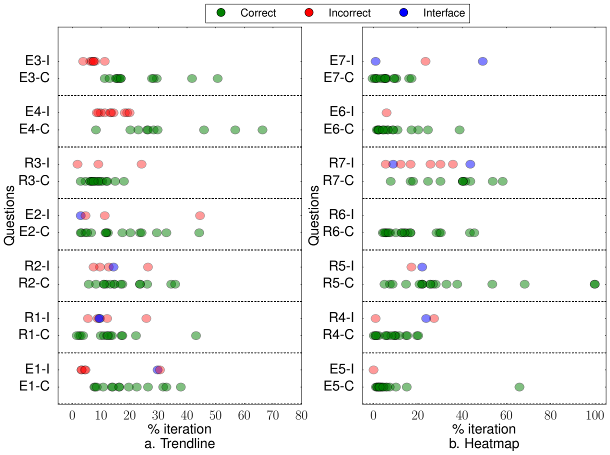


Figure 6: Per-question statistics for the iterations at which participants submitted answers for trendlines (l) and heatmaps (r).

6.3 Interview and Survey Phase

We now present a qualitative analysis of the participants' perceptions based on their interview and survey responses.

Summary: Participants were highly confident (confidence = 8.5 out of 10) when they submitted answers for both visualization types. Some participants, however, suggested providing explicit guarantees to further boost their confidence. Both trendline and heatmap visualizations were highly interpretable.

Interview. We conducted semi-structured interviews to gauge our participants' motivations for stopping the visualization at a certain iteration, and their confidence of their answers. The main motivations for terminating a visualization were the emergence of obvious extrema ($N = 5$), gaining sufficient confidence in an answer ($N = 10$), or both ($N = 5$). When asked to rate their confidence at the time of submission on a scale of 1 to 10 (10 being the most confident), most participants rated their confidence very high (μ = 8.5 and σ = 1.03 out of 10). However, some participants ($N = 4$) indicated that they would have continued until the final iteration if they were freely exploring the data (and not in an assessment setting). If they were pressed for time or the dataset was very large, they would choose to stop before the final visualization iteration, but likely at a later iteration than the iteration they stopped at in our quiz phase. One of those participants (P8) mentioned that providing an explicit guarantee along with the visualizations would further increase the confidence level when making decisions.

Survey. The survey consisted of seven Likert scale questions to measure the interpretability of the visualizations and the usability of IncVisage. The heatmap and trendline visualizations received similar average ratings (out of 5) for interpretability (heatmap: μ = 4.45; σ = 0.51; trendline: μ = 4.50; σ = 0.83) and satisfaction levels (heatmap: μ = 4.55; σ = 0.60; trendline: μ = 4.50; σ = 0.83).

Limitations and Future Work. We identified some limitations and possible future directions. Our approach to approximate visualizations relies heavily on the smoothness of the data; noise and outliers in the dataset impede generating a useful approximation quickly. As highlighted in Section 7.2, when the value of the point of interest and its neighbor(s) is very close, IncVisage might choose to reveal that point at later iterations. As a result, any user looking to find quick insights may select an incorrect sub-optimal point. IncVisage currently does not offer any explicit guarantee of an answer, which was pointed out as a limitation of the tool by one of the participants (P8).

Approach	Trendline				Heatmap			
	Extrema Based Questions		Range Based Questions		Extrema Based Questions		Range Based Questions	
	Accuracy	Time (sec)	Accuracy	Time (sec)	Accuracy	Time (sec)	Accuracy	Time (sec)
INCVISAGE	**94.55%**	**25.0638**	**62.50%**	**22.0822**	**83.47%**	**31.6012**	**97.50%**	34.7992
OLA	45.83%	26.4022	52.50%	27.3125	79.13%	**31.3846**	95%	**25.4782**

Table 3: Overall Accuracy and Submission Time Statistics Per Question Category

The limitations of the user study fall into three categories: ambiguity in quiz questions, interface control, and limited variance in the participant demographics. From the interface perspective, participants suggested adding axes to the snapshots for easier comparison between iterations, adding zooming capabilities, and options for downloading the data summary. Finally, our participant pool demographics do not match the demographics of the general audience intended for this tool. Future studies will reach a larger, more diverse audience.

7. EVALUATION: DECISION MAKING

Previously, we evaluated the interpretability of INCVISAGE, compared trendlines to heatmaps, and qualitatively studied how users felt about the inherent uncertainty. We now compare INCVISAGE with traditional online-aggregation-like [19] approaches (*OLA*) that depict approximations of visualizations as samples are taken (as in the first and third row of Figure 1). Specifically, does INCVISAGE lead to faster and/or more accurate decision making?

7.1 Study Design and Participants

Our study design was similar to our previous study, with four phases, an introduction, a quiz phase, an interview for qualitative data, and finally a closing survey. We introduced the INCVISAGE approach as the "grouped value" approach, and the *OLA* approach as the "single value" approach. We focus on the quiz phase; other findings and more details can be found in our technical report [2].

We recruited 20 participants. Our participants included 12 graduate students (4 in computer science), 5 undergraduate students, one researcher, and two university employees. All of the participants had experience with data analysis tools.

Quiz Phase Design. In designing the quiz phase, we used the flight dataset (details in Section 5), with 20 questions in total—10 on trendlines and 10 on heatmaps. In each case, five questions were reserved for INCVISAGE, and five for *OLA*. We used the same categorizations of questions as in our first study—range-based and extrema-based. As before, we randomized the order of the tools, and the order of the questions.

The Scoring Function. As in Section 6, a score was provided to the user as they answered questions. The score was computed as a product $S = P \cdot A$, where P corresponded to how quickly the user made a decision, and A to the accuracy. The formulae for A were similar to the previous study. Instead of setting P to be proportional to the number of remaining iterations, here, in order to allow the scores to be comparable between *OLA* and INCVISAGE, we set P to be $\frac{T-t}{T}$, where T is the time taken to compute the visualization by scanning the entire dataset, while t is the time taken by the user.

7.2 Quantitative Analysis of the Quiz Phase

In discussing the participants' quiz performance, we investigate both their accuracy (using A above) as well as answer submission time for both INCVISAGE and *OLA*.

Summary: For trendlines, INCVISAGE exhibits a 62.52% higher accuracy than *OLA* for both question types, while also reducing the submission time by 10.83%. For heatmaps, INCVISAGE exhibits 4.05% higher accuracy than *OLA* for both question types. The submission time for range-based questions with INCVISAGE is higher than *OLA*.

Accuracy and Submission Time Statistics. Table 3 summarizes the overall accuracy and the submission times for both INCVISAGE and *OLA*. For trendlines, INCVISAGE outperformed *OLA* in terms of both accuracy and submission times. For extrema based questions, the overall accuracy of INCVISAGE was almost double than that of *OLA*. The submission times were also lower for INCVISAGE for both types of questions. Overall, users are able to make *faster* and *more accurate* decisions using INCVISAGE than *OLA*. There is a dip in the overall accuracy for the range based questions for both approaches. Since the accuracy of the range based questions was either 0 or 1, any incorrect submission incurred a higher penalty, thereby reducing overall accuracy.

For heatmaps, INCVISAGE exhibited better accuracy than *OLA*—in particular, an improvement of 4.05%. For extrema based questions, the submission times were almost equal. However, for range based questions submissions with INCVISAGE took longer than *OLA*. We found that when using INCVISAGE with heatmaps, participants waited for the larger blocks to break up in order to compute averages over ranges, thereby increasing submission times but providing more accurate answers. As it turns out, the initial (larger) blocks in INCVISAGE do provide average estimates across ranges and could have been used to provide answers to the questions quickly. The unfamiliarity with incremental heatmap visualizations, and heatmaps in general, contributed to this delay. In hindsight, we could have educated the participants more about how to draw conclusions from the INCVISAGE heatmaps and this may have reduced submission times. In our technical report, we dig into these issues at a per-question level.

7.3 Interview and Survey Phase

Due to space limitations, here, we only present a summary of the results of the interview and survey phase, along with takeaway quotes; detailed results can be found in the technical report [2].

Overall, among 20 participants, the majority ($N = 12$) preferred the INCVISAGE ('grouped value') representation over the *OLA* ('single value') representation ($N = 6$) of visualizations while two participants equally preferred both approaches. When using INCVISAGE, participants reported that they were able to interpret the initial high level approximations to identify specific regions of interest and eventually found the answer after waiting a bit, with more confidence. On the other hand, they described *OLA* as unstable and difficult to interpret. One of the participants (P14) said the following—"For (INCVISAGE), it was easier to know when I wanted to stop because I had the overall idea first. And then I was just waiting to get the precise answer because I knew it was coming. . . . With (*OLA*), it was a shot in the dark where I see a little bit where it is, I would wait to see if it stays as the answer." Another participant (P15) expressed similar observations—"With (*OLA*), there was just so much going on I was like 'OK, where am I trying to focus on . . . versus (INCVISAGE), it started out really simple and then became more complex to be able to show the differences." The same participant also preferred the aesthetics of INCVISAGE—"I preferred the grouped (INCVISAGE), because it made it easier to kind of narrow down at least the range . . . Versus with the single value (*OLA*) . . . if you picked the wrong one, you were very wrong, potentially."

Summary: Participants were reasonably confident when they submitted answers for both visualization types using INCVISAGE. The trendline visualization using *OLA* was unstable and had low interpretability that resulted in participants submitting answers with low confidence. Majority of the participants preferred the INCVISAGE representations for both visualizations.

8. RELATED WORK

In this section, we review papers from multiple research areas and explain how they relate to INCVISAGE.

Approximate Query Processing (AQP). AQP schemes can operate online, i.e., select samples on the fly, and offline, i.e, select samples prior to queries being issued. Among online schemes, certain approaches respect a predefined accuracy constraint for computing certain fixed aggregates without indices [22, 23], and with indexes [18, 34]. The objectives and techniques are quite different from that of incrementally improving visualizations. Offline AQP systems [10, 9, 12, 17] operate on precomputed samples. Unlike these approaches, INCVISAGE deals with ad-hoc visualizations.

Approximate Visualization Algorithms. We have already discussed IFOCUS [30], PFunkH [11] and ExploreSample [50] in the introduction section. Concurrent work has shown that analysts are willing to use approximate visualizations in real data exploration scenarios [37]. This work introduces an optimistic visualization system that allows users to explore approximate visualizations and verify the results of any visualization they feel uncertain about at a later time. INCVISAGE's approach can be combined with this approach, since verification of decisions made using approximate visualizations may be valuable (and thus the two directions provide orthogonal and complementary benefits).

Incremental Visualization. We have already discussed Online aggregation [19] and sampleAction [16] in the introduction section. Recent user studies by Zgraggen et al. [51] demonstrate that an *OLA*-style system outperforms one-shot computation of the visualization in terms of number of insights gained—they do not contribute any new algorithms, however. In our work, we additionally demonstrate that INCVISAGE reduces the number of user mistakes made in decision making compared to *OLA*.

Bertolotto et al. [13] uses a multiresolution spatial maps construction technique [39] to compute representations of spatial maps at various resolutions offline to reduce time taken for storage and therefore data transfer at each resolution, while preserving geographical features of interest including the size and interaction of points, lines and shapes—this is therefore a offline data compression approach for a fixed spatial map. INCVISAGE, on the other hand, uses sampling to compute the k-increments of visualizations online and can support ad-hoc queries.

Visualization Tools. In recent years, several interactive visualization tools have been introduced [41, 45, 29, 43, 40]. The algorithms provided in this paper can be incorporated in these tools so that users can quickly identify key features of data.

Scalable Visualizations. A number of recent tools support scalable visualization generation [35, 28, 33] by precomputing and storing aggregates—this can be prohibitive on datasets with many attributes. INCVISAGE on the other hand, reveals features of visualizations in the order of prominence for arbitrary queries.

Approximation of Distributions. Approximating a data distribution by histograms has also been studied previously [8, 24, 26]. These methods do not sample iteratively from groups—they operate in a one-shot manner, and focus only on COUNT queries. Donjerkovic et al. [15] maintains histograms over evolving data, once again for COUNT queries.

9. CONCLUSIONS

We introduced the notion of incrementally improving visualizations and demonstrated that our incremental visualization tool, INCVISAGE, helps users gain insights and make decisions quickly. On very large datasets, INCVISAGE is able to achieve a 46× speedup relative to *SCAN* in revealing the first 10 salient features of a visualization with suitable error guarantees that are comparable to a dynamic programming approach, but without a high computational overhead. Our user studies demonstrate that users chose to trade accuracy for time to make rapid decisions, that too at higher accuracy than traditional approaches. There are a number of interesting future directions, such as modeling and displaying the degree of uncertainty, along with a wider range of operations (e.g. pausing at segment level or group level), and alternative views (e.g., overlaying incremental visualizations and traditional approaches). Finally, gaining a better understanding of the sorts of decisions for which one-shot approaches and incremental visualization approaches are appropriate is a promising future direction.

10. REFERENCES

[1] Intel sensor dataset. http://db.csail.mit.edu/labdata/labdata.html.

[2] I've seen enough: Incrementally improving visualizations to support rapid decision making. Technical report. data-people.cs.illinois.edu/incvisage.pdf.

[3] Microsoft's power bi hits 5m subscribers, adds deeper excel integration. http://www.pcworld.com/article/3047083/. Accessed: 05-22-2016.

[4] Nyc taxi dataset. http://publish.illinois.edu/dbwork/open-data/.

[5] Tableau q2 earnings: Impressive growth in customer base and revenues. http://www.forbes.com/sites/greatspeculations/2015/07/31/tableau-q2-earnings-impressive-growth-in-customer-base-and-revenues.

[6] Us flight dataset. http://stat-computing.org/dataexpo/2009/the-data.html.

[7] Wunderground weather dataset. https://www.wunderground.com/.

[8] J. Acharya et al. Fast and near-optimal algorithms for approximating distributions by histograms. In *PODS*, pages 249–263. ACM, 2015.

[9] S. Acharya et al. The aqua approximate query answering system. In *SIGMOD Rec.*, volume 28, pages 574–576. ACM, 1999.

[10] S. Agarwal et al. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42. ACM, 2013.

[11] D. Alabi et al. Pfunk-h: Approximate query processing using perceptual models. In *HILDA Workshop*, pages 10:1–10:6. ACM, 2016.

[12] B. Babcock et al. Dynamic sample selection for approximate query processing. In *SIGMOD Conf.*, pages 539–550. ACM, 2003.

[13] M. Bertolotto et al. Progressive vector transmission. In *Proceedings of the 7th ACM international symposium on Advances in geographic information systems*, pages 152–157. ACM, 1999.

[14] M. Correll et al. Error bars considered harmful: Exploring alternate encodings for mean and error. *IEEE TVCG*, 20(12):2142–2151, 2014.

[15] D. Donjerkovic et al. Dynamic histograms: Capturing evolving data sets. In *ICDE'00*, pages 86–86. IEEE Computer Society Press; 1998, 2000.

[16] D. Fisher et al. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *CHI'12*, pages 1673–1682. ACM, 2012.

[17] M. N. Garofalakis. Approximate query processing: Taming the terabytes. In *VLDB*, 2001.

[18] P. J. Haas et al. Selectivity and cost estimation for joins based on random sampling. *Journal of Computer and System Sciences*, 52(3):550–569, 1996.

[19] J. M. Hellerstein et al. Online aggregation. *SIGMOD Rec.*, 26(2), 1997.

[20] J. M. Hellerstein et al. Interactive data analysis: The control project. *Computer*, 32(8):51–59, 1999.

[21] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[22] W.-C. Hou et al. Statistical estimators for relational algebra expressions. In *PODS*, pages 276–287. ACM, 1988.

[23] W.-C. Hou et al. Processing aggregate relational queries with hard time constraints. In *SIGMOD Rec.*, volume 18, pages 68–77. ACM, 1989.

[24] P. Indyk et al. Approximating and testing k-histogram distributions in sub-linear time. In *PODS*, pages 15–22. ACM, 2012.

[25] F. Jackson. The basic gamma-function and the elliptic functions. *Proceedings of the Royal Society of London.*, 76(508):127–144, 1905.

[26] H. V. Jagadish et al. Optimal histograms with quality guarantees. In *VLDB*, volume 98, pages 24–27, 1998.

[27] U. Jugel et al. M4: a visualization-oriented time series data aggregation. *VLDB Endow.*, 7(10):797–808, 2014.

[28] S. Kandel et al. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *AVI*, pages 547–554. ACM, 2012.

[29] A. Key et al. Vizdeck: self-organizing dashboards for visual analytics. In *SIGMOD Conf.*, pages 681–684. ACM, 2012.

[30] A. Kim et al. Rapid sampling for visualizations with ordering guarantees. *VLDB*, 8(5):521–532, 2015.

[31] A. Kim et al. Speedy browsing and sampling with needletail. Technical report, 2016. https://arxiv.org/abs/1611.04705.

[32] N. Koudas. Space efficient bitmap indexing. In *CIKM*, pages 194–201, 2000.

[33] L. Lins et al. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE TVCG*, 19(12):2456–2465, 2013.

[34] R. J. Lipton et al. Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, 116(1):195–226, 1993.

[35] Z. Liu et al. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[36] Z. Liu et al. The effects of interactive latency on exploratory visual analysis. *IEEE TVCG*, 20(12):2122–2131, 2014.

[37] D. Moritz et al. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. In *CHI*, 2017.

[38] S. G. Perlman. System and method for rendering graphics and video on a display, June 26 2007. US Patent 7,236,204.

[39] E. Puppo et al. Towards a formal model for multiresolution spatial maps. In *Advances in Spatial Databases*, pages 152–169. Springer, 1995.

[40] T. Siddiqui et al. Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. *VLDB Endowment*, 10(4):457–468, 2016.

[41] C. Stolte et al. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE TVCG*, 8(1):52–65, 2002.

[42] K. Stromberg. *Probability for analysts*. CRC Press, 1994.

[43] M. Vartak et al. Seedb: efficient data-driven visualization recommendations to support visual analytics. *VLDB Endow.*, 8(13):2182–2193, 2015.

[44] Weisstein, Eric W, Wolfram Research, Inc. Euler-mascheroni constant. 2002.

[45] R. Wesley et al. An analytic data engine for visualization in tableau. In *SIGMOD Conf.*, pages 1185–1194. ACM, 2011.

[46] M. B. Wilk et al. Probability plotting methods for the analysis for the analysis of data. *Biometrika*, 55(1):1–17, 1968.

[47] A. P. Witkin. Scale-space filtering: A new approach to multi-scale description. In *ICASSP*, volume 9, pages 150–153. IEEE, 1984.

[48] K. Wu et al. Optimizing bitmap indices with efficient compression. *ACM Transactions on Database Systems (TODS)*, 31(1):1–38, 2006.

[49] K. Wu et al. Analyses of multi-level and multi-component compressed bitmap indexes. *ACM Transactions on Database Systems (TODS)*, 35(1):2, 2010.

[50] Y. Wu et al. Efficient evaluation of object-centric exploration queries for visualization. *VLDB*, 8(12):1752–1763, 2015.

[51] E. Zgraggen et al. How progressive visualizations affect exploratory analysis. *IEEE TVCG*, 2016.