Distributed and Parallel Databases (2021) 39:665–710
https://doi.org/10.1007/s10619-020-07315-w

Stratified random sampling from streaming and stored data

Trong Duc Nguyen[1] · Ming-Hung Shih[1] · Divesh Srivastava[2] · Srikanta Tirthapura[1] · Bojian Xu[3]

Accepted: 9 October 2020 / Published online: 23 October 2020

Abstract

Stratified random sampling (SRS) is a widely used sampling technique for approximate query processing. We consider SRS on continuously arriving data streams and statically stored data sets. We present a tight lower bound showing that any streaming algorithm for SRS over the entire stream must have, in the worst case, a variance that is $\Omega(r)$ factor away from the optimal, where r is the number of strata. We present `S-VOILA`, a practical streaming algorithm for SRS over the entire stream that is *locally variance-optimal*. We prove that any sliding window-based streaming SRS needs a workspace of $\Omega(rM \log W)$ in the worst case, to maintain a variance-optimal SRS of size M, where W is the number of elements in the sliding window. Due to the inherent high workspace needs for sliding window-based SRS, we present `SW-VOILA`, a multi-layer practical sampling algorithm that uses only $O(M)$ workspace but can maintain an SRS of size close to M in practice over a sliding window. Experiments show that both `S-VOILA` and `SW-VOILA` result in a variance that is typically close to their optimal offline counterparts, which was given the entire input beforehand. We also present `VOILA`, a variance-optimal offline algorithm for stratified random sampling. `VOILA` is a strict generalization of the well-known *Neyman allocation*, which is optimal only under the assumption that each stratum is abundant. Experiments show that `VOILA` can have significantly smaller variance (1.4x to 50x) than Neyman allocation on real-world data.

Keywords Stratified random sampling · Stream sampling · Sliding window sampling · Neyman allocation

A preliminary version of this work appears in [1].

✉ Trong Duc Nguyen
trong@iastate.edu

[1] Iowa State University, Ames, USA

[2] AT&T - Research, Austin, USA

[3] Eastern Washington University, Cheney, USA

1 Introduction

Random sampling is a widely-used method for data analysis, and features prominently in the toolbox of virtually every approximate query processing system. The power of random sampling lies in its generality. For many important classes of queries, an approximate answer whose error is small in a statistical sense can be efficiently obtained through executing the query over an appropriately derived random sample. Sampling operators are part of all major database products, *e.g.,* Oracle, Microsoft SQL Server, and IBM Db2. The simplest method for random sampling is uniform random sampling, where each element from the entire data (the "population") is chosen with the same probability. Uniform random sampling may however lead to a high variance in estimation. For instance, consider a population $D = \{1, 2, 4, 2, 1, 1050, 1000, 1200, 1300\}$, and suppose we wanted to estimate the population mean. A uniform random sample of size two leads to an estimate with a variance of approximately 1.6×10^5.

An alternative sampling method is *stratified random sampling* (SRS), where the population is partitioned into subgroups called "strata". From within each stratum, uniform random sampling is used to select a per-stratum sample. All per-stratum samples are combined to derive the "stratified random sample". Suppose that the population is divided into two strata, one with elements $\{1, 2, 4, 2, 1\}$ and the other with elements $\{1000, 1050, 1200, 1300\}$. A stratified random sample of size two that chooses one element from each stratum yields an estimate with variance 2.47×10^3, much smaller than a uniform random sample of the same size.

SRS provides the flexibility to emphasize some strata over others through controlling the allocation of sample sizes; for instance, a stratum with a high standard deviation can be given a larger allocation than another stratum with a smaller standard deviation. In the above example, if we desire a stratified sample of size three, it is best to allocate a smaller sample of size one to the first stratum and a larger sample size of two to the second stratum, since the standard deviation of the second stratum is higher. Doing so, the variance of estimate of the population mean further reduces to approximately 1.23×10^3. The strength of SRS is that a stratified random sample can be used to answer queries not only for aggregates such as the mean, and sum of the entire population, but also of subsets of the population defined by selection predicates that are provided at query time. SRS has been used widely in database systems for approximate query processing [2–4].

A problem with handling large streaming data is that current methods for SRS are predominantly offline methods that assume all data is available before sampling starts. As a result, systems that rely on SRS (*e.g.,* [3–5]) cannot easily adapt to new data and have to recompute stratified random samples from scratch, as more data arrives. With the advent of streaming data warehouses such as Tidalrace [6], it is imperative to have methods for SRS that work on dynamic data streams, and maintain stratified random samples in an incremental manner.

We address the shortcoming of current methods through a study of SRS on streaming data. The difficulty of SRS on streaming data is that there are two logical processes simultaneously at work. One is sample size allocation, which

allocates samples among the different strata in a manner that minimizes the variance of an estimate. The second is the actual sampling of elements from within each stratum. While each of these two steps, sample size allocation and sampling, can be done individually in a streaming fashion, it is far more challenging to do them simultaneously. We present lower bounds as well as algorithms for the task of maintaining a stratified random sample on a data stream. The quality of a stratified random sample is measured using the variance of an estimate of a population statistic, computed using the sample.

1.1 Our contributions

Streaming lower bounds We present a lower bound showing that in the worst case, any streaming algorithm for SRS over the entire stream that uses a memory of M records must have a variance that is $\Omega(r)$ away from the variance of the optimal offline algorithm that uses the same memory of M records, where r is the number of strata. We show that this lower bound is tight, by construction. We also present a lower bound showing that any streaming algorithm for SRS over a timestamp sliding window needs a workspace of $\Omega(rM \log W)$ in order to maintain a variance-optimal SRS of M records from the sliding window, where W is an upper bound of the number of elements in the sliding window.

Practical streaming algorithm for SRS over the entire stream We present S-VOILA (Streaming Variance OptImaL Allocation), a streaming algorithm for SRS that is locally variance-optimal. Upon receiving new elements, it (re-)allocates sample sizes among strata so as to obtain the smallest variance among all possible re-allocations. S-VOILA can also deal with the case when a minibatch of multiple data items is seen at a time, as in systems such as Spark streaming [7]. Re-allocations made by S-VOILA are locally optimal with respect to the entire minibatch, and the quality of re-allocations improve as the minibatch size increases. Since S-VOILA can deal with minibatches of varying sizes, it is well-suited to real-world streams that may have bursty arrivals.

Practical streaming algorithm for SRS over a sliding window We present SW-VOILA, an algorithm for sampling from a timestamp sliding window of the most recent elements in a stream. This algorithm uses only $O(M)$ workspace, which is much smaller than the $\Omega(rM \log W)$ lower bound, but can maintain an SRS whose size is close to M records in practice. The variance of the sample produced by this algorithm is also empirically demonstrated to be close to the one obtained by an optimal offline algorithm that takes multiple passes through data in the window. SW-VOILA takes a novel multi-layer sampling approach that is composed of multiple carefully designed samples of different sample rates. These multiple layers of samples allow the dynamic changes of their sample rates in order to adapt to the progression of the sliding window, and can be collectively used to produce a good quality SRS from the sliding window.

Variance-optimal sample size reduction The streaming algorithms (S-VOILA and SW-VOILA) re-allocate sample sizes based on a novel method for reducing the size of an existing stratified random sample down to a desired target size in a

variance-optimal manner. This novel technique for sample size reduction may be of independent interest in other tasks, e.g., sub-sampling from a given stratified random sample.

Variance optimal offline SRS We present the first offline algorithm for variance-optimal SRS. Our algorithm VOILA computes an allocation with provably optimal variance among all possible allocations of sample sizes to strata. The well known **Neyman Allocation** [8] (NeyAlloc), which originated from the statistics literature, assumes that each stratum has an abundance of data to choose from. However, this assumption may not hold in databases, since each stratum is a subset of a database table, and the size of a stratum may be small. VOILA does not make such assumptions, and computes a variance optimal allocation no matter how large/small the sizes of the strata. Hence, VOILA is a strict generalization of NeyAlloc. In addition, VOILA does not make any assumption on how data is stratified.

Experimental evaluation We present a detailed experimental evaluation using real and synthetic data, considering both the quality of sample and accuracy of query answers using the sample. In our experimental study, we found that:

(a) The variance of S-VOILA is typically close to that of the optimal offline algorithm VOILA, and the allocation of S-VOILA also closely track that of VOILA. S-VOILA improves significantly upon prior work [9]. The variance of S-VOILA improves as the size of the minibatch increases, and a minibatch of size 100 provides most of the benefits of S-VOILA.

(b) Samples produced using S-VOILA yield accurate answers to a range of queries that involve a selection followed by aggregation, where the selection predicate is provided at query time, and the aggregation function can be one of sum, average, and standard deviation.[1]

(c) SW-VOILA can produce sliding window-based SRS of size close to the given memory (workspace)—at least as large as 97% of the workspace size. The variance of the sample produced by SW-VOILA is smaller than those from Senate (the memory budget is shared evenly among all strata), Reservoir and S-VOILA in the setting of sliding windows, and is close to that of VOILA, the optimal offline algorithm for SRS

(d) In the offline setting, VOILA can have significantly smaller variance than Senate and NeyAlloc.

1.2 Related work

Streaming stratified random sampling in the online setting can be viewed as weight-based reservoir sampling where the weight of each stream element depends on the stratification of the stream and the statstics of the strata. Since the weight of a stream element changes dynamically (even after it has been observed), prior work on weighted reservoir sampling [10] does not apply as their work assumes the weight of each element to be unchanged. Meng [11] considered streaming SRS

[1] Note that a query for the variance or standard deviation of data is distinct from the variance or standard deviation of an estimate.

using population-based allocation and thus does not achieve the goal of optimality in variance that we are targeting. Al-Kateb *et al.* [9, 12] considered streaming SRS using power allocation, based on their prior work on adaptive reservoir sampling [13]. Their work does not guarantee the uniformity of each sample in the SRS. They also did not consider the case of bounded strata. Prior work on streaming SRS neither considers provable guarantees on the quality of the resulting samples, nor lower bounds for streaming SRS, like we do here.

Stratified random sampling on static data has been studied for long and can be done using two scans of the data set. The first scan is to learn the statistics of each stratum and use them to decide the sample size allocation. Example allocation policies that have been developed by the statstics community include the Neyman allocation [8] and power allocation [14]. The second scan of the data set is to draw the actual sample for each stratum using the allocated the sample size, and this can be done via the classic reservoir sampling [15] or weighted reservoir sampling [10]. A recent interesting work by Lang et al. [16] consider machine learning methods for determining the per-item probability of inclusion in a sample. This work is meant for static data, and can be viewed as a version of weighted random sampling where the weights are learnt using a query workload.

Stratified random sampling has had a long history of usage in the database community for approximate query processing. The heuristic-based congressional sampling from [17], which is also used in their system Aqua [2], is a hybrid of population-based allocation ("house") and fixed allocation ("senate") with the goal of serving a collection of group-by queries, while our work `VOILA` targets a different goal that is to optimize the variance of the whole population estimate with a theoretical guarantee. The "small group sampling" from [18] draws uniform random samples from big groups while keeping the entire data set of small groups in their preprocessing phase that construct a family of multiple samples. For each query, they dynamically select and combine an appropriate subset of the pre-computed samples so as to serve the query well. Chaudhuri et al. [5] formulate the approximate query processing as an optimization problem using SRS for a query workload. SRS for low selectivity queries was studied in [19]. BlinkDB [3] uses SRS as the fundamental tool to allow users to have the trade-off between query accuracy and query response time. Quickr [4] is a query-time sampling system that use the first pass of the data to generate samples to serve a query that needs multiple passes over the data. Using the random samples generated during the first pass helps speed up the subsequence query processing. A recent work "Sample+Seek" [20] uses a combination of measure-biased sampling and an index to help with low- selectivity predicates. All of these prior work [2–5, 17–19] that use SRS for approximate query processing have assumed static data and it is not clear how to adapt them in a streaming setting. However, with the emergence of data stream processing systems [21] and data stream warehousing systems [6], it is important to devise methods for streaming SRS with quality guarantees.

Non-stratified random sampling has been widely used in approximate query processing on both static and streaming data [22–26]. The reservoir sampling [27, 28] algorithm for uniform sampling from a stream has been known for decades, and many variants and generalizations have been considered, such as

weight-based sampling [10, 29], insertion and deletion of elements [30], distinct sampling [31], sampling from a sliding window [30, 32–34], time-decayed sampling [35, 36], and distributed streaming sampling [37–40].

2 Problem statement

Stratified sampling can be viewed as being composed of three parts—stratification, sample allocation, and sampling. Stratification is a partitioning of the universe into a number of disjoint strata. Equivalently, it is the assignment of each data element to a unique stratum. In database applications, stratification is usually a pre-defined function of one or more attributes of the data [22]. For example, the works of Chaudhuri et al. [5] and Agarwal et al. [3] on approximate query answering stratify tuples in a database table based on the set of selection predicates in the query workload that the tuple satisfies, and the work of Kandula et al. [4] on approximate query answering stratify rows of a table using the group ids derived from a group-by query. Note that our methods do not assume that stratification is performed in any specific manner, and work regardless of the method used to stratify data.

Our work considers sample allocation, the partitioning of the available memory budget of M samples among the different strata. In streaming SRS, the allocation needs to be continuously re-adjusted as more data arrives, and the characteristics of different strata change. In offline sampling, allocation needs to be done only once, after knowing the data in its entirety.

The final sampling step considers each stratum and chooses the assigned number of samples uniformly at random. In offline stratified sampling, the sampling step can be performed in a second pass through the data using reservoir sampling on the subset of elements belonging to each stratum, after a first pass has determined the sample size allocation. In the case of streaming sampling, the sampling step needs to occur simultaneously with sample (re-)allocation, which may change allocations to different strata over time.

Variance-optimal allocation The quality of a stratified random sample is measured through the variance of an estimate that is derived using the sample. Consider a data stream $R = \{v_1, v_2, \ldots, v_n\}$ of current size n, whose elements are stratified into r strata, numbered $1, 2, \ldots, r$. Let n_i denote the number of elements in stratum i. For each $i = 1 \ldots r$, let S_i be a uniform random sample of size s_i drawn without replacement from stratum i. Let $\mathbb{S} = \{S_1, S_2, \ldots, S_n\}$ denote the stratified random sample. The sample mean of each per-stratum sample S_i is: $\bar{y}_i = \frac{\sum_{v \in S_i} v}{s_i}$. The population mean of R, μ_R can be estimated as: $\bar{y} = \frac{\sum_{i=1}^{r} n_i \bar{y}_i}{n}$. It can be shown that the expectation of $\bar{y}$ equals μ_R. Given a memory budget of $M \leq n$ elements to store all the samples, so that $\sum_i s_i = M$, the following question of *variance-optimal allocation* of sample sizes has been considered in prior work [8]: *How to split the memory budget M among the s_i's to minimize the variance of* $\bar{y}$? The variance of $\bar{y}$ can be computed as follows (e.g. see Theorem 5.3 in [22]):

$$V = V(\bar{y}) = \frac{1}{n^2}\sum_{i=1}^{r} n_i(n_i - s_i)\frac{\sigma_i^2}{s_i} = \frac{1}{n^2}\sum_{i=1}^{r}\frac{n_i^2\sigma_i^2}{s_i} - \frac{1}{n^2}\sum_{i=1}^{r} n_i\sigma_i^2 \tag{1}$$

While the theory around SRS in both statistics and database communities has used the variance of the population mean as a minimization metric, variance-optimal SRS is useful for other types of queries as well, including predicate-based selection queries, sum queries across a subset of the strata, queries for the variance, and combinations of such queries [3, 5]—also see Sect. 7.

`NeyAlloc` *for Abundant Strata* Prior studies on variance-optimal allocation have primarily considered static data. Additionally, they assume that every stratum has a very large volume of data, so that there is no restriction on the size of a sample that can be chosen from this stratum. This may not be true for the scenario of databases. Especially in a streaming context, each stratum starts out with very little data. Given a collection of data elements R, we say that a stratum i is *abundant* if $n_i \geq M \cdot (n_i\sigma_i)/\left(\sum_{j=1}^{r} n_j\sigma_j\right)$. Otherwise, the stratum i is said to be *bounded*. Under the assumption that each stratum is abundant, the popularly used "Neyman Allocation" `NeyAlloc` [8, 22] minimizes the variance V, and allocates a sample size for stratum i as

$$M_i = M \cdot \frac{n_i\sigma_i}{\sum_{j=1}^{r} n_j\sigma_j} \tag{2}$$

We note that `NeyAlloc` is no longer optimal if one or more strata are bounded. Our methods of sample size reduction and online (`S-VOILA` and `SW-VOILA`) and offline (`VOILA`) algorithms do not have this restriction and work under the general case whether or not strata are bounded.

Our solution to streaming SRS consists of two parts—sample size re-allocation, and per-stratum random sampling. Both parts execute continuously and in an interleaved manner. Sample size re-allocation is achieved using a reduction to a "sample size reduction" in a variance-optimal manner. Given a stratified random sample $\mathbb{S}_1$ of size larger than a target M, sample size reduction seeks to find a stratified sample $\mathbb{S}_2$ of size M that is a subset of $\mathbb{S}_1$ such that the variance of $\mathbb{S}_2$ is as small as possible.

Roadmap In Sect. 3, we consider streaming SRS over the entire stream. We present a tight lower bound for this challenge, followed by `S-VOILA`, a practical and *locally optimal* algorithm for streaming SRS over the entire steram. In Sect. 4, we prove a lower bound for streaming SRS over a timestamp sliding window, followed by `SW-VOILA`, a streaming algorithm that uses much less space than the lower bound but can produce an SRS whose size is close to the entire memory cost of the algorithm, making the algorithm a practical solution for streaming SRS over a sliding window. Both `S-VOILA` and `SW-VOILA` use as a subroutine a novel variance-optimal sample size reduction technique that we describe in Sect. 5. We start with `SingleElementSSR` for reducing the size of the sample by one element, followed by a general algorithm `SSR` for reducing the size by $\beta \geq 1$ elements. We then present an algorithm `MultiElementSSR` with a faster runtime. We then present `VOILA`, the optimal offline algorithm for SRS in Sect. 6. We present an

Table 1 Notations used throughout the paper

Variable	Definition
$\mathcal{R}$	Data stream, $\|\mathcal{R}\| = n$
n	Number of elements in the data stream
M	Sample size, $M < n$
B	Minibatch of size b elements
r	Number of strata
i	A single stratum, $i = 1 \ldots r$
n_i, μ_i, σ_i	Size, mean, and standard deviation of stratum i
$\mathcal{R}_i$	Stream of elements from stratum i, $\|\mathcal{R}_i\| = n_i$
S_i	Sample for stratum i, $\|S_i\| = s_i$
M_i	Allocated sample size for stratum i
θ_i	Acceptance threshold of selecting an element into S_i
t	Clock time
$W(t)$	Number of elements in the window at t
$W_i(t)$	Number of elements for stratum i in the window at t
$n_i(t), \mu_i(t), \sigma_i(t)$	Size, mean, and standard deviation of stratum i in the window at t
S_i^j	Sample of stratum i in the layer j
θ_i^j	Acceptance threshold of selecting an element into S_i^j
p_i^j	Timestamp of the oldest element of stratum i in the layer j

experimental study of our algorithms in Sect. 7, followed by a conclusion in Sect. 8. Table 1 summarizes the set of notations we use in this paper.

3 Streaming SRS over an infinite window

We now consider SRS from an entire data stream, whose elements are arriving continuously. As more elements are seen, the allocations as well as samples need to be dynamically adjusted. We first note there is a simple two-pass streaming algorithm with optimal variance that uses $O(k + r)$ space, where k is the desired sample size and r the number of strata. In the first pass, the size, mean, and standard deviations of each stratum are computed using $O(r)$ space, constant space for each stratum. At the end of the first pass, the allocations to different strata are computed using an optimal offline algorithm, say `VOILA` (Sect. 6). In the second pass, since the desired sample sizes are known for each stratum, samples are computed using reservoir sampling within the substream of elements belonging to each stratum. The above two-pass algorithm *cannot be* converted into a one-pass algorithm. The difficulty is that as more elements are seen, allocations to different strata may change, and the sampling rate within a stratum cannot in general be (immediately) dynamically adjusted in order to satisfy variance optimality. We first show a lower bound that it is in general not possible for any streaming algorithm to have optimal variance compared with an offline algorithm that is given the same memory.

3.1 A lower bound for streaming SRS over an infinite window

Given a data stream $\mathcal{R}$ with elements belonging to r strata, and a memory budget of M elements, let V^* denote the optimal sample variance that can be achieved by an offline algorithm for SRS that may make multiple passes through data. Clearly, the sample produced by any streaming algorithm must have variance that is either V^* or greater. Suppose a stratified random sample R is computed by a streaming algorithm using memory of M elements. Let $V(R)$ denote the variance of this sample. For $\alpha \geq 1$, we say *R is an SRS with multiplicative error of α, if: (1) the sample within each stratum in R is chosen uniformly from all elements in the stratum, and (2) $V(R) \leq \alpha \cdot V^*$.*

Theorem 1 *Any streaming algorithm for maintaining an SRS over a stream with r strata using a memory of M elements must, in the worst case, result in a stratified random sample with a multiplicative error $\Omega(r)$.*

The idea in the proof is to construct an input stream with r strata where the variance of different strata are the same until a certain point in time, at which the variance of a single stratum starts increasing to a high value—a variance-optimal SRS will respond by increasing the allocation to this stratum. However, we show that a streaming algorithm is unable to do so quickly. Though a streaming algorithm may compute the variance-optimal allocation to different strata in an online manner, it cannot actually maintain these dynamically sized samples using limited memory.

Proof Consider an input stream where for each $i = 1 \ldots r$, the ith stratum consists of elements in the range $[i, i+1)$. The stream so far has the following elements. For each $i, 1 \leq i \leq r$, there are $(\alpha - 1)$ copies of element i and one copy of $(i + \varepsilon)$ where $\varepsilon = 1/(r-1)$ and $\alpha \geq 3$. After observing these elements, for stratum i we have $n_i = \alpha$, $\mu_i = \left(i + \frac{\varepsilon}{\alpha}\right)$, and it can be verified that $\sigma_i = \frac{\sqrt{\alpha-1}}{\alpha}\varepsilon$.

Since the total memory budget is M, at least one stratum (say, Stratum 1) has a sample size no more than M/r. Suppose an element of value $(2 - \varepsilon)$ arrives next. This element belongs to stratum 1. Let n_1', μ_1', and σ_1' denote the new size, mean, and standard deviation of stratum 1 after this element arrives. We have $n_1' = \alpha + 1$ and $\mu_1' = 1 + \frac{1}{\alpha+1}$. It can be verified that $\sigma_1' = \sqrt{\frac{\varepsilon^2 + (1-\varepsilon)^2 - \frac{1}{\alpha+1}}{\alpha+1}}$. It follows that:

$$(\alpha+1)\sqrt{\frac{\frac{1}{2} - \frac{1}{\alpha+1}}{\alpha+1}} \leq n_1'\sigma_1' \leq (\alpha+1)\sqrt{\frac{1 - \frac{1}{\alpha+1}}{\alpha+1}} \tag{3}$$

$$\implies \frac{\sqrt{\alpha}}{2} \leq n_1'\sigma_1' \leq \sqrt{\alpha} \qquad \text{(Note: } \alpha > 2\text{)} \tag{4}$$

In 3, the left inequality stands when $\varepsilon = 1/2$ and the right inequality stands when $\varepsilon = 0$ or 1. We also have: $\sum_{i=2}^{r} n_i\sigma_i = (r-1)\alpha\frac{\sqrt{\alpha-1}}{\alpha}\varepsilon = \sqrt{\alpha - 1}$, where we have used $\varepsilon = \frac{1}{r-1}$. Thus,

$$\frac{\sqrt{\alpha}}{2} \leq \sum_{i=2}^{r} n_i \sigma_i \leq \sqrt{\alpha} \qquad \text{(Note: } \alpha > 2\text{)} \tag{5}$$

Let V denote the sample variance of $\mathcal{A}$ after observing the stream of $(r\alpha + 1)$ elements. Let V^* denote the smallest sample variance possible with a stratified random sample of size M on this data. Let $\Delta = \left(n'_1{}^2\sigma'^2_1 + \sum_{i=2}^{r} n_i^2\sigma_i^2\right)/n^2$.

We observe that after processing these $(r\alpha + 1)$ elements, the sample size $s_1 \leq M/r + 1$. Using this fact and the definition of sample variance in Eq. 1:

$$\begin{aligned} V =& \frac{1}{n^2}\left(\frac{n'_1{}^2\sigma'^2_1}{s_1} + \sum_{i=2}^{r}\frac{n_i^2\sigma_i^2}{s_i}\right) - \Delta \geq \frac{1}{n^2}\left(\frac{n'_1{}^2\sigma'^2_1}{\frac{M}{r}+1} + \sum_{i=2}^{r}\frac{n_i^2\sigma_i^2}{\frac{M}{r-1}}\right) - \Delta \\ \geq& \frac{1}{n^2}\left(\frac{\alpha/4}{\frac{M}{r}+1} + \sum_{i=2}^{r}\frac{(\alpha-1)\varepsilon^2}{M/(r-1)}\right) - \Delta = \frac{1}{n^2}\left(\frac{\alpha/4}{\frac{M}{r}+1} + \frac{\alpha-1}{M}\right) - \Delta \end{aligned}$$

On the other hand, the smallest sample variance V^* is achieved by using Neyman allocation. By Inequalities 4 and 5, we know that if Neyman allocation is for the current stream of $r\alpha + 1$ elements, stratum 1 uses at least $M/3$ memory space, whereas all other strata equally share at least $M/3$ elements since all $n_i\sigma_i$ are equal for $i = 2, 3, \ldots, r$. Using these observations into Equation 1:

$$\begin{aligned} V^* \leq& \frac{1}{n^2}\left(\frac{n'_1{}^2\sigma'^2_1}{M/3} + \sum_{i=2}^{r}\frac{n_i^2\sigma_i^2}{M/(3(r-1))}\right) - \Delta \\ \leq& \frac{1}{n^2}\left(\frac{\alpha}{M/3} + \sum_{i=2}^{r}\frac{(\alpha-1)\varepsilon^2}{M/(3(r-1))}\right) - \Delta = \left(\frac{1}{n^2}\frac{6\alpha-3}{M}\right) - \Delta \end{aligned}$$

Since $\Delta \geq 0$ and $M > r$, we have: $\frac{V}{V^*} \geq \frac{V+\Delta}{V^*+\Delta} = \Omega(r)$. □

We note that the above lower bound is tight (up to constant factors). Consider the algorithm which always allocates M/r memory to each of r strata that have been observed so far. Using the formula for the variance of an SRS in Equation 1, we can verify that this algorithm has a variance within an $O(r)$ multiplicative factor of the optimal. While theoretically such an algorithm (which we call the "senate" algorithm due to allocating every stratum the same resources) meets the worst case lower bound, it performs poorly in practice, since it treats all strata equally, irrespective of their volume or variance (see the experiments section).

3.2 S-VOILA: streaming algorithm for SRS over an infinite window

We now present a streaming algorithm `S-VOILA` that can maintain a stratified random sample on a stream with a good (though not optimal) variance. Given a memory budget of M items, `S-VOILA` maintains a SRS of size M with the following properties: (1) the samples within each stratum are chosen uniformly from all the stream elements seen

in the stratum so far, (2) the sizes of samples allocated to different strata adapt to new stream elements by making "locally optimal" decisions that lead to the best allocations given the new stream elements. S-VOILA conceptually has to solve two problems. One is sample size re-allocation among strata, and the second is uniform sampling within each stratum. Let $\mathcal{R}$ denote the stream observed so far, and $\mathcal{R}_i$ the elements in $\mathcal{R}$ that belong to stratum i.

We first consider sample size re-allocation. Suppose due to the addition of new elements, the stream went from $\mathcal{R}^1$ to $\mathcal{R}^2$, and suppose that the stratified random sample at $\mathcal{R}^1$ allocated sample sizes to strata in a specific manner, S^1. Due to the new elements, the sizes and variances of different strata change, and as a result, the optimal allocation of samples in $\mathcal{R}^2$ may be different from the previous allocation S^1. Our approach is to first add new elements to the sample, and then re-allocate sample sizes using a "variance-optimal sample size reduction" optimization framework. *Given a current allocation of sample sizes to different strata, suppose new elements are added to the sample, causing it to exceed a memory threshold M. What is a way to reduce the current sample to a sample of size M such that the variance of the new sample is as small as possible?* In Sect. 5, we present algorithms for sample size reduction.

The second issue is to maintain a uniform random sample S_i of $\mathcal{R}_i$ when s_i, the size of the sample is changing. A decrease in an allocation to s_i can be handled easily, through discarding elements from the current sample S_i until the desired sample size is reached. What if we need to increase the allocation to stratum i? If we simply start sampling new elements according to the higher allocation to S_i, then recent elements in the stream will be favored over the older ones, and the sample within stratum i is no longer uniformly chosen. In order to ensure that S_i is always chosen uniformly at random from $\mathcal{R}_i$, newly arriving elements in $\mathcal{R}_i$ need to be held to the same sampling threshold as older elements, even if the allotted sample size s_i increases. S-VOILA resolves this issue in the following manner. An arriving element from $\mathcal{R}_i$ is assigned a random "key" drawn uniformly from the interval (0, 1). The sample is maintained using the following invariant: *S_i is the set of s_i elements with the smallest keys among all elements so far in $\mathcal{R}_i$*. It is easy to verify that this is indeed a uniform sample drawn without replacement from $\mathcal{R}_i$. The consequence of this strategy is that if we desire to increase the allocation to stratum i, it may not be accomplished immediately, since a newly arriving element in $\mathcal{R}_i$ may not be assigned a key that meets this sampling threshold. Instead, the algorithm has to wait until it receives an element in $\mathcal{R}_i$ whose assigned key is small enough. To ensure the above invariant, the algorithm maintains for each stratum i a variable θ_i that tracks the smallest key of an element in $\mathcal{R}_i$ that is not currently included in S_i. If an arriving element in $\mathcal{R}_i$ has a key that is smaller than θ_i, it is included within S_i; otherwise, it is not.

Algorithm 1: S-VOILA: Initialization

Input: M – total sample size, r – number of strata.
// S_i is the sample for stratum i, and $\mathcal{R}_i$ is the substream of elements from Stratum i
Load the first M stream elements in memory, and partition them into per-stratum samples, $S_1, S_2, \ldots, S_r$, such that S_i consists of (e, d) tuples from stratum i, where e is the element, d is the key of the element, chosen independently and uniformly at random from $(0, 1)$.
For each stratum i, compute n_i, σ_i. Initialize $\theta_i \leftarrow 1$ // θ_i tracks the smallest key among all elements in $\mathcal{R}_i$ not selected in S_i

Algorithm 2: S-VOILA: Process a new minibatch B of b elements. Note that b need not be known in advance, and can vary from one minibatch to the other.

```
1  for each e ∈ B do
2      Let α = α(e) denote the stratum of e
3      Update n_α and σ_α    // per-stratum mean and std. dev. maintained in a
         streaming manner
4      Assign a random key d(e) ∈ (0, 1) to element e
5      if d(e) < θ_α then                                  // element e is sampled
6          S_α ← {e} ⋃ S_α;
   /* Variance-optimal reduction down to M elements                          */
7  if |S| − M = 1 then                          // faster for evicting 1 element
8      ℓ ← SingleElementSSR(M)
9      x ← arg max_{x∈S_ℓ} d(x)
10     S_ℓ ← S_ℓ \ {x}
11     θ_ℓ = d(x)
12 else if |S| − M > 1 then
13     L ← MultiElementSSR(M)
14     for i = 1 ... r do                                  // Actual element evictions
15         if L[i] < |S_i| then
16             Delete |S_i| − L[i] elements from S_i with the largest keys
17             θ_i ← smallest key discarded from S_i
```

Here is a short example to demonstrate the idea of S-VOILA. Suppose a new element e arrives, we update the statistics of its corresponding stratum, and a random key will be generated to determine if e is selected into the sample. If e is selected and the sample is full, we evict one element from sample using the variance-optimal sample size reduction technique from Sect. 5. Otherwise, we discard e. Similar process could be applied to multiple new elements. Algorithms 1 and 2 respectively describe the initialization and insertion of a minibatch of elements. S-VOILA supports the insertion of a minibatch of any size $b > 0$, where b can change from one minibatch to another. As b increases, we can expect S-VOILA to have a lower variance, since its decisions are based on greater amount of data. Lines 1–6 make one pass through the minibatch to update the mean and standard

deviations of the strata, and store selected elements into the per-stratum samples. If $\beta > 0$ elements from the minibatch get selected into the sample, in order to balance the memory budget at M, β elements need to be evicted from the stratified random sample.

A sample size reduction algorithm takes a current allocation to a stratified random sample, the statistics (volume, mean, and variance) of different strata, and a target sample size M, and returns the final allocation whose total size is M. For the special case of evicting one element, we can use the faster algorithm `SingleElementSSR`; otherwise, we can use `MultiElementSSR`. Lemma 1 shows that the sample maintained by `S-VOILA` within each stratum is a uniform random sample, showing this is a valid stratified sample, and Lemma 2 presents the time complexity analysis of `S-VOILA`.

Lemma 1 *For each $i = 1, 2, \ldots, r$ sample S_i maintained by* `S-VOILA` *(Algorithm 2) is selected uniformly at random without replacement from stratum R_i.*

Proof First, note that each S_i is selected from $\mathcal{R}_i$ without replacement, because each element of $\mathcal{R}_i$ is selected into S_i no more than once. Next, we prove the uniformity of S_i. In case $|S_i| = n_i$, all elements of $\mathcal{R}_i$ are in S_i. In case $|S_i| < n_i$, S_i contains the $|S_i|$ elements with the smallest keys from stratum $\mathcal{R}_i$, because: (1) Anytime an element is discarded from S_i, it is the element of the largest key in the sample. (2) If another element e with key $d(e)$ enters later, it cannot be inserted into S_i unless $d(e)$ is smaller than all other keys discarded so far. Because the keys of elements are assigned randomly, each element has a chance of $|S_i|/n_i$ to be selected into S_i. Therefore, S_i is a uniform random sample from $\mathcal{R}_i$ without replacement. □

Lemma 2 *If the minibatch size $b = 1$, then the worst-case time cost of* `S-VOILA` *for processing an element is $O(r)$. The expected time for processing an element belonging to stratum α is $O(1 + r \cdot s_\alpha / n_\alpha)$, which is $O(1)$ when $r \cdot s_\alpha = O(n_\alpha)$. If $b > 1$, then the worst-case time cost of* `S-VOILA` *for processing a minibatch is $O(r \log r + b)$.*

Proof $b = 1$: The worst case happens when the single new element from belonging to stratum α gets selected into S_α. In that case, we need to reduce the stratified random sample size by one via `SingleElementSSR`, which takes $O(r)$ time. The probability that the new element is selected into S_α is equal to s_α / n_α, so the expected time follows.

$b > 1$: The time cost for Lines 1–6 is $O(b)$. The time cost for Lines 1–6 is $O(r \log r + \beta)$. So the total time cost is $O(b) + O(r \log r + \beta) = O(r \log r + b)$. □

We can expect `S-VOILA` to have an amortized per-item processing time of $O(1)$ in many circumstances. When $b = 1$: After observing enough stream elements from stratum α, such that $r \cdot s_\alpha = O(n_\alpha)$, the expected processing time of an element becomes $O(1)$. Even if certain strata have a very low frequency, the expected time cost for processing a single element is still expected to be $O(1)$, because elements from an infrequent stratum α are unlikely to appear in the

minibatch. When $b > 1$: The per-element amortized time cost of S-VOILA is $O(1)$, when the minibatch size $b = \Omega(r \log r)$.

4 Streaming SRS over a sliding window

We consider maintaining an SRS from a timestamped sliding window. Let t denote the clock time that starts from 0 and increases by 1 at every clock tick. A sliding window of *length* Δ consists of the most recent elements observed in the data stream during the clock time $[t - \Delta, t]$. Let $W_i(t)$ denote the number of elements that belong to stratum i in the window at time t. We call $W(t) = \sum_{i=1}^{r} W_i(t)$ the *size* of the sliding window at time t. We consider the window size W to be very large and thus storing the entire window is infeasible. Similar to S-VOILA in Sect. 3.2, the algorithm to maintain an SRS over sliding window consists of two parts: sample size re-allocation and sampling, which are interleaved with each other.

For re-allocating sample sizes, we need the statistics of each stratum within the sliding window. The required memory space to calculate the exact mean and variance over a sliding window is linear to W [41], which is not practical with a large window size. We adopt existing works on approximations of the mean [41, 42] and variance [43, 44].

To maintain the uniformity within each stratum, we use the mechanism presented in Sect. 3.2, which assigns each arriving element a random key chosen uniformly in (0, 1). Each stratum maintains a threshold for selecting new elements, which is the smallest key that has *not* been selected. However, an exact tracking of that smallest key for each stratum requires memory linear in W [41]. In Sect. 4.2, we propose an approximation of that smallest key for each stratum.

Selecting new samples may cause the sample size to exceed the allocated memory. We use our variance-optimal sample size reduction technique, *i.e.,* Algorithm MultiElementSSR, to reallocate sample sizes to each stratum such that the total size of SRS is controlled under the memory budget. Those strata whose sample size shall be reduced are sub-sampled by evicting the elements of the largest keys from the stratum's current sample.

4.1 A lower bound for streaming SRS over a sliding window

We first prove that to maintain a variance-optimal SRS over a timestamped sliding window, there exists a lower bound of memory space which is larger than the size of SRS. Let $n_i(t)$, $\mu_i(t)$, and $\sigma_i(t)$ denote the size, mean, and standard deviation of the stratum i in the sliding window at time t, respectively. The allocated sample size for stratum i is denoted as $M_i(t)$ and can be calculated using Equation 2. For the clarity of context, we omit t when using these notations for this section.

Theorem 2 *Any streaming algorithm that maintains an SRS of size M over a sliding window needs at least $\Omega(rM \log W)$ space in the expectation for the worst case, such that every S_i is a uniform random sample and every $s_i = M_i$.*

The idea for the proof shares a similar spirit from the proof for Theorem 1, but also considers the additional challenges in the sliding window-based random sample maintenance. The idea for the proof is to show that we can construct a stream such that each stratum i can potentially receive a new element that significantly increases the variance of stratum i, causing M_i, the memory allocation size for stratum i, to be close to M. That means, in order to ensure that $s_i = M_i$ is still maintained after receiving such a new element, s_i needed to be close to M before the new element arrives. Because every stratum can potentially receive such a new element, it becomes mandatory for every stratum to maintain a random sample of size close to M. Because the expected space cost for maintaining a uniform random sample of size M for a single stratum i over the sliding window is at least $\Omega(M \log W_i)$ ([34]), we can verify the correctness of the theorem.

Proof Consider an input stream where for each $i = 1 \ldots r$, the ith stratum consists of elements in the range $[i, i+1)$. At every clock time $t \geq \Delta$ up to the current clock time denoted as t_c, every stratum i has the following set of elements within the window of length Δ: There are $(\alpha - 1)$ copies of element i and one copy of $(i + \varepsilon)$, where $\varepsilon = 1/(r-1)$ and $\alpha \geq 3$. That is, at every clock time $t = \Delta, \Delta + 1, \ldots, t_c$, we have $W_i = \alpha = W/r$, $n_i(t) = \alpha$, $\mu_i(t) = \left(i + \frac{\varepsilon}{\alpha}\right)$, and $\sigma_i(t) = \frac{\sqrt{\alpha-1}}{\alpha}\varepsilon$, for every i.

For each stratum i, we call an element of value $(i + 1 - \epsilon)$ a *bad* element. Suppose at this current time t_c, we also receive a bad element for stratum 1. Let $n'_1(t_c)$, $\mu'_1(t_c)$, and $\sigma'_1(t_c)$ denote the new size, mean, and standard deviation of stratum 1 within the current window after the bad element arrives. It is easy to calculate that $n'_1(t_c) = \alpha + 1$, $\mu'_1(t_c) = 1 + \frac{1}{\alpha+1}$, and $\sigma'_1(t_c) = \sqrt{\frac{\varepsilon^2 + (1-\varepsilon)^2 - \frac{1}{\alpha+1}}{\alpha+1}}$. It follows that:

$$\begin{aligned} &(\alpha+1)\sqrt{\frac{\frac{1}{2} - \frac{1}{\alpha+1}}{\alpha+1}} \leq n'_1(t_c)\sigma'_1(t_c) \leq (\alpha+1)\sqrt{\frac{1 - \frac{1}{\alpha+1}}{\alpha+1}} \\ \Longrightarrow\ &\frac{\sqrt{\alpha}}{2} \leq n'_1(t_c)\sigma'_1(t_c) \leq \sqrt{\alpha} \end{aligned} \tag{6}$$

We also have: $\sum_{i=2}^{r} n_i(t_c)\sigma_i(t_c) = (r-1)\alpha\frac{\sqrt{\alpha-1}}{\alpha}\varepsilon = \sqrt{\alpha - 1}$. Thus,

$$\frac{\sqrt{\alpha}}{2} \leq \sum_{i=2}^{r} n_i(t_c)\sigma_i(t_c) \leq \sqrt{\alpha} \tag{7}$$

From Inequalities 6 and 7, we can observe that in a variance-optimal SRS, the memory allocation size for stratum 1 is as least $M/3$, i.e., $M_1 \geq M/3$. After Stratum 1 receives its bad element at time t_c, there is no way to ensure $s_1 = M_1 \geq M/3$ unless the sampler has maintained a sample S_1 whose size is at least $M/3$ before the bad element of Stratum 1 arrives.

Note that every stratum can receive its bad element at any clock time, in order to ensure $s_i = M_i$ for every stratum i at every clock time, we have to maintain a random sample S_i for every stratum i such that $s_i \geq M/3$ at every clock time. We also know from [34] that the expected workspace cost for maintaining a single

uniformly random sample of size M over a sliding window of size W is at least $\Omega(M \log W)$ in the worst case. Therefore, the expected workspace needs for maintaining a random sample of size $M/3$ for Stratum i over the sliding window is at least $\Omega(M \log W_i) = \Omega(M \log \alpha) = \Omega\big(M \log(W/r)\big)$ in the worst case. Adding all the workspaces needs by all strata, the expected workspace needs for maintaining a variance-optimal SRS over a sliding window is at least $\Omega\big(rM \log(W/r)\big) = \Omega(rM \log W)$ in the worst case. The last equality is because r is polynomial smaller than W. Otherwise, we can just cache the entire window and there is no need to design a sublinear-space streaming algorithm. □

This lower bound can indeed be matched (other than a constant factor) by a probabilistic upper bound for maintaining a variance-optimal SRS over the sliding window. It can be proved that the following algorithm can maintain a variance-optimal SRS over the sliding window with a high probability and it uses $O\big(M \sum_{i=1}^{r} \log(W_i)\big) = O(rM \log W)$ space in expectation: For each stratum i, we maintain $\lceil \log_2 W_i \rceil + 1$ buffers named as $\mathcal{B}_0^i, \mathcal{B}_1^i, \ldots, \mathcal{B}_{\lceil \log_2 W_i \rceil}^i$, where each buffer has size M. Every buffer $\mathcal{B}_j^i$ saves the M most recent elements that are selected uniformly at random from stratum i with probability $1/2^j$. When a query for an SRS arrives, for each stratum i, we pick the smallest $j \in \{0, 1, \ldots, \lceil \log_2 W_i \rceil\}$, such that buffer $\mathcal{B}_j^i$ is saving all the selected elements from the current window, and use buffer $\mathcal{B}_j^i$ to generate S_i for stratum i.

Due to the inherent high workspace needs as shown by the lower bound in Theorem 2, we next present a practical algorithm that uses a workspace of only $O(M)$ but can maintain a sliding window-based SRS whose size is very close to M, the target SRS size. Experimental results (Sect. 7) shows its quality in query answering is close to the optimum.

4.2 SW-VOILA: a practical algorithm for sliding window SRS

In the sliding window setting, elements collected in the sample will expire and be removed, leaving vacancies in the sample. Arbitrarily accepting newly arrived elements to fill out the vacancies is not a good approach since it breaks the uniformity within each stratum.

In order to keep the uniformity, we observe that the sample rate can only decrease in the infinite window setting. However, in the sliding window setting, it is possible to increase the sample rate while the uniformity is still maintained. Given two non-overlapped timestamp windows: $[t_1, t_2]$ and $[t_3, t_4]$, where $t_2 < t_3$. Both windows are uniform, at the sample rates of r_1 and r_2, respectively. It is possible to have $r_1 < r_2$, *i.e.,* the sample rate increases over time. However, if we simply increase the sample rate from r_1 to r_2, when the window slides between t_2 and t_3, the samples are nonuniform. It is nontrivial to keep the uniformity when the window is sliding between two frames.

We introduce `SW-VOILA`, an algorithm for maintaining a stratified random sample over the sliding window of a stream. `SW-VOILA` fully utilizes the given memory M over the sliding window by having the sampler rate increasing over time.

Meanwhile, SW-VOILA guarantees a sample with uniformity within each stratum at any time. The key insight of SW-VOILA is to maintain multiple layers of the sample with growing sample rates. The first layer is the base maintained in the same way as S-VOILA except the expired elements will be removed. The second layer (and onward if necessary) uses the leftover memory from the first layer to accept newly arrived elements at a higher sampling rate. While the uniformity of the first layer is always guaranteed, the second and upper layers serve as buffers to keep more elements which *potentially* can be used later.

Let θ_i^j denote the acceptance threshold of the layer j of the stratum i. For convenience, we set $\theta_i^0 = 0$. All other thresholds have initial value 1. Let p_i^j is the timestamp of the oldest element in layer j of stratum i, that is used to keep track of the window frame in which the layer collects its sample. Every element e is assigned a random key $d(e) \in (0, 1)$. e belongs to a layer k of its stratum i, such that $\theta_i^{k-1} \leq d(e) < \theta_i^k$. Let S denote the current sample set. $S_i \subset S$ is the subset of samples that belong to stratum i. We define:

$$S_i^j = \{e \in S_i \mid \theta_i^{j-1} \leq d(e) < \theta_i^j\}$$

is the subset of sample that belongs to the layer j of stratum i. The base sample $\bigcup_{i=1}^{r} S_i^1$ is returned to user when requested, as in Algorithm 4.

Algorithm 3: `SW-VOILA`: Process a new minibatch B, at the current timestamp $t(B)$

```
   /* S_i^j = {e ∈ S_i | θ_i^{j-1} ≤ d(e) < θ_i^j}: elements of the layer j of stratum i  */
1  S = {e ∈ S | t(e) + Δ ≥ t(B)}                          // Remove expired elements
2  for i = 1 ... r do
3      if p_i^2 + Δ ≤ t(B) then                  // Merge the first and second layers
4          p_i^j = p_i^{j+1}, ∀j
5          θ_i^j = θ_i^{j+1}, ∀j
6      Update n̂_i and σ̂_i
7  for e ∈ B do
8      Let α denote the stratum of e
9      Assign a random key d(e) ∈ (0, 1) to element e
10     k = min{j | d(e) < θ_α^j}
11     S = S ∪ {e}
12     if |S_α^k| = 1 and θ_α^k = 1.0 then
13         p_α^k = t(e)                          // New (highest) layer of stratum α
   /* Remove from an upper layer                                                    */
14 while |S| > M and Σ_{i,j>1} |S_i^j| > 0 do
15     {x, β, l} = arg max_{x∈S_β^l, l>1} d(x)
16     S = S \ {x}
17     θ_β^l = d(x)
18     θ_β^j = 1.0, j > l
   /* Remove from the base layer                                                    */
19 if |S| − M > 0 then
20     L ← MultiElementSSR(M)
21     for i = 1 ... r do
22         if L[i] < |S_i^1| then
23             Delete |S_i^1| − L[i] elements from S_i^1 with the largest keys
24             θ_i^1 ← smallest key discarded from S_i
25             θ_i^j = 1.0, j > 1
```

Algorithm 4: `SW-VOILA`: get stratified random sample that guarantee the uniformity within each stratum

```
1 return ⋃_{i=1}^r S_i^1
```

Figure 1 demonstrates an example of a stratum α with 4 layers of the sample. The base layer (green) is uniform since it contains all elements in the current window, whose random key is less than θ_α^1. Meanwhile, the combination of the first and second layers in the time frame from p_α^2 to the current timestamp is uniform as well. It is because the combination contains all the elements seen from p_α^2

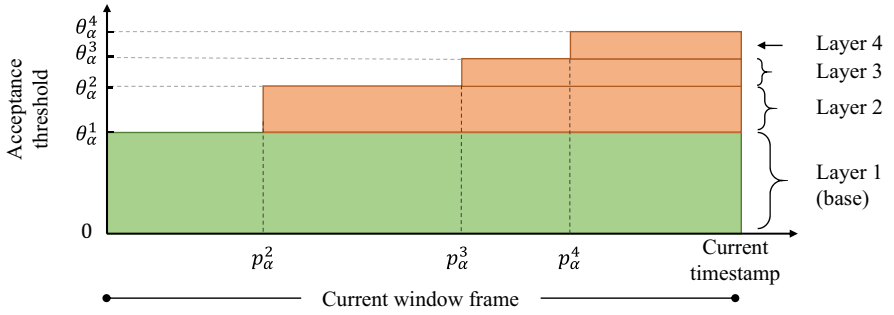


Fig. 1 Example of 4 layers of sample in stratum α. Both the acceptance thresholds and the starting times of layers are in order

until now, whose key is less than θ_α^2. Similarly, in the upper layers, the combination of samples within the corresponding time frames are uniform.

As the window slides (from left to right), p_α^2 eventually matches the starting time of the current window. That is the *merging point* where we can safely merge layer 2 into the base sample since the combination of the first and second layer becomes uniform in the current window. We mark the other layer's index down by one, *i.e.,* layer 3 becomes the new layer 2.

`SW-VOILA` supports the general case of timestamp-based sliding window, where the window length is Δ. The counting-based sliding window can be considered as a special case, where the timestamp increases by 1 at receiving every element and thus $W = \Delta$. Algorithm 3 handles a new minibatch B received from the stream. The size of B is unknown in advance and can vary from one minibatch to another. As the window slides, we first remove all expired elements (Line 1). When a stratum α has the second layer fit the current time frame, *i.e.,* its starting point p_α^2 is at least a window size away from current timestamp, the second layer is merged into the base sample (layer 1). The upper layers are re-indexed (Lines 3–5). Each new element e in B is assigned a random key and added to its corresponding layer (Lines 7–11). If e is the first element of its layer, we mark down the timestamp of e as the starting time p_α^k of this layer (Lines 12–13). As new elements are received and expired elements are removed, we update the frequency and standard deviation of each stratum over the current window, that are needed to run the algorithm, using an existing work [41–44].

The most crucial part of the algorithm is to keep the memory usage within the limit, *i.e.,* eliminating elements when the size of S exceeds M. To preserve the serving sample, we remove elements from the base layer only when all upper layers are empty. If they are not empty, the elements with the largest key in the upper layers is removed (line 14–18). Once all the upper layers are empty but the used memory still larger than the limitation, the `SW-VOILA` algorithm behaves similarly to `S-VOILA`. It uses `MultiElementSSR` to choose elements to be removed such that it minimizes the increase of the variance of the sample (Lines 19–25). When an element is removed, its layer's threshold is set to the key of the removed item. Note that, the higher layers of the same stratum are all

empty. In other words, we remove items from the highest non-empty layer of the chosen stratum.

Theorem 3 *For each $i = 1, 2, \ldots, r$, sample S_i^1 returned by* SW-VOILA *(Algorithm 4) is selected uniformly at random without replacement from the current window of the stratum i.*

Proof Each element in S_i^1 is selected without replacement from the current window because a new element e is chosen exactly once into its layer S_i^j of the stratum it belongs to. Next, we prove the uniformity of S_i^1 due to the following reasoning:

1. Before any merging, S_1^1 is maintained as a S-VOILA sample. By Lemma 1, it is uniform.
2. The second layer S_i^2 is merged into the first layer S_i^1 only if its starting point p_i^2 is at least a window-size older than the current timestamp. On the other hand, all the expired elements are removed from the sample. Thus both S_i^1 and S_i^2 contain elements from the current window. Since S_i^1 has all the elements from stratum i whom the key is less than θ_i^1, while all elements of stratum i with their key in range $[\theta_i^1, \theta_i^2)$ belong to S_i^2, the combination $S_i^1 \cup S_i^2$ contains all the elements from stratum i, within the current window, whose key is less than θ_i^2. Because the keys of elements are assigned randomly between (0, 1), each element has a chance of θ_i^2 to belong to $S_i^1 \cup S_i^2$. Therefore, $S_i^1 = S_i^1 \cup S_i^2$ is a uniform random sample of stratum i at this merging point.
3. After merged, S_1^1 is maintained as a S-VOILA sample again, until the next merging point. Thus it is uniform during the time between 2 merging points.

□

5 Variance-optimal sample size reduction

Suppose it is necessary to reduce a stratified random sample (SRS) of total size M to an SRS of total size $M' < M$. This will need to reduce the size of the samples of one or more strata in the SRS. Since the sample sizes are reduced, the variance of the resulting estimate will increase. We consider the task of *variance-optimal sample size reduction* (VOR), *i.e.,* how to partition the reduction in sample size among the different strata in such a way that the increase in the variance is minimized. Note that once the new sample size for a given stratum is known, it is easy to subsample the stratum to the target sample size.

Consider Equation 1 for the variance of an estimate derived from the stratified random sample. Note that, for a given data set, a change in the sample sizes of different strata s_i does not affect the parameters n, n_i, and σ_i. VOR can be formulated as the solution to the following non-linear program.

$$\text{Minimize} \sum_{i=1}^{r} \frac{n_i^2 \sigma_i^2}{s'_i} \tag{8}$$

subject to constraints:

$$\sum_{i=1}^{r} s'_i = M' \qquad \text{and} \qquad 0 \le s'_i \le s_i \text{ for each } i = 1, 2, \ldots, r, \tag{9}$$

In the rest of this section, we present efficient approaches for computing the `VOR`.

5.1 Sample size reduction by one element

We first present an efficient algorithm for the case where the size of a stratified random sample is reduced by one element. An example application of this case is in designing a streaming algorithm for SRS, when stream items arrive one at a time. The task is to choose a stratum i (and discard a random element from the stratum) such that after reducing the sample size s_i by one, the increase in variance V (Eq. 1) is the smallest.

Lemma 3 *When required to reduce the size of a stratified random sample by one, the increase in variance of the estimated population mean is minimized if we reduce the size of S_ℓ by one, where $\ell = \arg\min_i \left\{ \frac{n_i^2 \sigma_i^2}{s_i(s_i-1)} \,\middle|\, 1 \le i \le r \right\}$.*

Proof For $i = 1, 2, \ldots, r$, let Δ_i denote the increase of the variance of the population mean estimate if s_i is reduced by one. Using Equation 1, we have

$$\Delta_i = \frac{n_i^2 \sigma_i^2}{n^2} \frac{1}{s_i(s_i - 1)}.$$

It is obvious that in order to minimize the increase of the variance, we shall reduce s_ℓ by one, where $\ell = \arg\min_i \left\{ \frac{n_i^2 \sigma_i^2}{s_i(s_i-1)} \,\middle|\, 1 \le i \le r \right\}$. □

Lemma 3 naturally makes sure that strata are not missing from the sample by giving the infinity penalty to remove the last element of a stratum, which prevents the sample of every stratum from being empty. In the case where we have multiple choices for ℓ using Lemma 3, although it is practically rare, we choose the one where the current sample size s_ℓ is the largest. Algorithm `SingleElementSSR` for reducing the sample by a single element is a direct implementation of the condition stated in Lemma 3. It is straightforward to observe this can be done in $O(r)$ time.

Algorithm 5: `SingleElementSSR()`: Variance-Optimal Sample Size Reduction by One

```
return arg min_i { (n_i^2 σ_i^2) / (s_i(s_i − 1)) | 1 ≤ i ≤ r }  /* The id of the stratum whose sample
    size shall be reduced by one. */
```

5.2 Sample size reduction by $\beta \geq 1$ elements

We now consider the general case, where the sample needs to be reduced by $\beta \geq 1$ elements. A possible solution idea is to repeatedly apply the one-element reduction algorithm (Algorithm `SingleElementSSR`from Sect. 5.1) β times. At each iteration, we choose and discard a single element from a stratum, hoping the overall variance increases caused by the discartion of β elements is minimized. However, this greedy approach may not yield a sample with the smallest variance. On the other hand, an exhaustive search of all possible evictions is not feasible either, since the number of possible ways to partition a reduction of size β among r strata is $\binom{\beta+r-1}{r-1}$, which can be very large. For instance, if $r = 10$, this is $\Theta(\beta^{10})$. We now present efficient approaches to `VOR`. We first present a recursive algorithm, followed by a faster iteration-based method. Before presenting the algorithm, we present the following useful characterization of a variance-optimal reduction.

Definition 1 We say that stratum i is *oversized* under memory budget M, if its allocated sample size $s_i > M_i$. Otherwise, we say that stratum i is *not oversized*.

Lemma 4 *Suppose that E is the set of β elements that are to be evicted from a stratified random sample such that the variance V after eviction is the smallest possible. Then, each element in E must be from a stratum whose current sample size is oversized under the new memory budget $M' = M - \beta$.*

Proof We use proof by contradiction. Suppose one of the evicted elements is deleted from a sample S_α such that the sample size s_α is not oversized under the new memory budget. Because the order of the eviction of the β elements does not impact the final variance, suppose that element e is evicted after the other $\beta - 1$ evictions have happened. Let s_α denote the size of sample S_α at the moment t right after the first $\beta - 1$ evictions and before evicting e. The increase in variance caused by evicting an element from S_α is

$$\Delta = \frac{1}{n^2}\left(\frac{n_\alpha^2\sigma_\alpha^2}{s_\alpha(s_\alpha - 1)}\right) = \left(\frac{\sum_{i=1}^{r} n_i\sigma_i}{nM'}\right)^2 \frac{M'^2_\alpha}{s_\alpha(s_\alpha - 1)}$$
$$> \left(\frac{\sum_{i=1}^{r} n_i\sigma_i}{nM'}\right)^2$$

where $M'_\alpha = M' \frac{n_\alpha \sigma_\alpha}{\sum_{i=1}^{r} n_i \sigma_i}$. The last inequality is due to the fact that S_α is not oversized under budget M' at time t, i.e., $s_\alpha \leq M'_\alpha$.

Note that an oversized sample exists at time t, since there are a total of $M' + 1$ elements in the stratified random sample at time t, and the memory target is M'. Instead of evicting e, if we choose to evict another element e' from an oversized sample $S_{\alpha'}$, the resulting increase in variance will be:

$$\begin{aligned}\Delta' &= \frac{1}{n^2}\left(\frac{n_{\alpha'}^2 \sigma_{\alpha'}^2}{s_{\alpha'}(s_{\alpha'}-1)}\right) = \left(\frac{\sum_{i=1}^{r} n_i \sigma_i}{nM'}\right)^2 \frac{M'^2_{\alpha'}}{s_{\alpha'}(s_{\alpha'}-1)} \\ &< \left(\frac{\sum_{i=1}^{r} n_i \sigma_i}{nM'}\right)^2\end{aligned}$$

where $M'_{\alpha'} = M' \frac{n_{\alpha'} \sigma_{\alpha'}}{\sum_{i=1}^{r} n_i \sigma_i}$ The last inequality is due to the fact that $S_{\alpha'}$ is oversized under budget M' at time t, i.e., $s_{\alpha'} > M'_{\alpha'}$. Because $\Delta' < \Delta$, at time t, evicting e' from $S_{\alpha'}$ leads to a lower variance than evicting e from S_α. This is a contradiction to the assumption that evicting e leads to the smallest variance, and completes the proof. □

Lemma 4 implies that it is only necessary to reduce samples that are oversized under the target memory budget M'. Samples that are not oversized can be given their current allocation, even under the new memory target M'. Our algorithm based on this observation first allocates sizes to the samples that are not oversized. The remaining memory now needs to be allocated among the oversized samples. We note that this can again be viewed as a sample size reduction problem, while focusing on a smaller set of (oversized) samples, and accomplish it using a recursive call under a reduced memory budget; See Lemma 5 for a formal statement of this idea. The base case for this recursion is when all samples under consideration are oversized, in which case we can simply use `NeyAlloc` under the reduced memory budget M' (Observation 1). Our algorithm `SSR` is shown in Algorithm 6.

Algorithm 6: SSR($\mathcal{A}, M, \mathcal{L}$): Variance-optimal sample size reduction

Input: $\mathcal{A}$ – set of strata under consideration.
M – target sample size for all strata in $\mathcal{A}$.
Output: For $i \in \mathcal{A}$, $\mathcal{L}[i]$ is the final size of sample for stratum i.

```
O ← ∅   // oversized samples
for j ∈ A do
    M_j ← M · n_j σ_j / Σ_{t∈A} n_t σ_t                    // Neyman allocation
    if (s_j > M_j) then O ← O ∪ {j}
    else L[j] ← s_j                                        // Keep current allocation
if O = A then
    for j ∈ A do L[j] ← M_j        // All samples oversized. Recursion stops
else
    SSR(O, M − Σ_{j∈A−O} s_j, L)      // Recurse on O, under remaining budget
```

Let $\mathbb{S} = \{S_1, S_2, \ldots, S_r\}$ be the current stratified random sample. Let $\mathcal{A}$ denote the set of all strata under consideration, initialized to $\{1, 2, \ldots, r\}$. Let $\mathcal{O}$ denote the set of oversized samples, under target memory budget for $\mathbb{S}$, and $\mathcal{U} = \mathbb{S} - \mathcal{O}$ denote the collection of samples that are not oversized. When the context is clear, we use $\mathcal{O}$, $\mathcal{U}$, and $\mathcal{A}$ to refer to the set of stratum identifiers as well as the set of samples corresponding to these identifiers.

Lemma 5 *A variance-optimal eviction of β elements from $\mathbb{S}$ under memory budget M' requires a variance-optimal eviction of β elements from $\mathcal{O}$ under memory budget $M' - \sum_{j \in \mathcal{U}} s_j$.*

Proof Recall that s'_i denotes the final size of sample S_i after β elements are evicted. Referring to the variance V from Eq. 1, we know a variance-optimal sample size reduction of β elements from $\mathbb{S}$ under memory budget M' requires minimization of

$$\sum_{i \in \mathcal{A}} \frac{n_i^2 \sigma_i^2}{s'_i} - \sum_{\in \mathcal{A}} \frac{n_i^2 \sigma_i^2}{s_i} \tag{10}$$

By Lemma 4, we know $s_i = s'_i$ for all $i \in \mathcal{U}$. Hence, minimizing Formula 10 is equivalent to minimizing

$$\sum_{i=\mathcal{O}} \frac{n_i^2 \sigma_i^2}{s'_i} - \sum_{i \in \mathcal{O}} \frac{n_i^2 \sigma_i^2}{s_i} \tag{11}$$

The minimization of Formula 11 is exactly the result obtained from a variance-optimal sample size reduction of β elements from oversized samples under the new memory budget $M' - \sum_{i \in \mathcal{U}} s_i$. □

Observation 1 *In the case every sample in the* stratified random sample *is oversized under target memory M', i.e., $\mathbb{S} = \mathcal{O}$, the variance-optimal reduction reduces the size of each sample $S_i \in \mathbb{S}$ to M'_i under the new memory budget M'.*

Table 2 An example of variance-optimal sample size reduction from 400×10^6 down to 200×10^6

i	1	2	3	4	5	6
$n_i\sigma_i$ $(\times 10^9)$	10	8	30	20	8	24
s_i $(\times 10^6)$	15	50	50	45	60	180
Round 1 M_i $(\times 10^6)$	20	< 50	60	< 45	< 60	< 180
Round 2 M_i $(\times 10^6)$	–	< 50	–	45	< 60	< 180
Round 3 M_i $(\times 10^6)$	–	18	–	–	18	54
s'_i $(\times 10^6)$	15	18	50	45	18	54

The following theorem summarizes the correctness and time complexity of Algorithm SSR.

Theorem 4 *Algorithm 6* (SSR) *finds a variance-optimal reduction of the stratified random sample $\mathcal{A}$ under new memory budget M. The worst-case time of* SSR *is $O(r^2)$, where r is the number of strata.*

Proof Correctness follows from Lemmas 4–5 and Observation 1. The worst-case time happens when each recursive call sees only one stratum that is not oversized. In such a case, the time of all recursions of SSR on a stratified random sample across r strata is: $O(r + (r-1) + \ldots + 1) = O(r^2)$. □

An Example (Table 2). Suppose we have 6 strata with their statistics ($n_i\sigma_i$) and current sample sizes (s_i) showin in Table 2 using a total size of $\sum_{i=1}^{6} s_i = 400$. Suppose that we wish to reduce the sample size down to 200 by reducing each s_i to the target sample size s'_i. The computation involves a sequence of recursive rounds. In the initial round, we allocate 200 samples among all 6 strata using Neyman allocation. Strata 1 and 3 turn out to be not oversized ($M_1 \geq s_1$, $M_3 \geq s_3$), and therefore we set $s'_1 = s_1$ and $s'_3 = s_3$. In Round 2, we exclude strata 1 and 3 from consideration, and the available memory budget which now becomes $200 - 15 - 50 = 135$. This is allocated among strata 2, 4, 5, and 6 using Neyman allocation. Stratum 4 is not oversized ($M_4 \geq s_4$) and therefore we set $s'_4 = s_4$. At the next round 3, we further exclude stratum 4 from consideration, and the available memory budget now becomes $135 - 45 = 90$. When this is allocated among the remaining strata, it turns out that all of them are oversized ($M_i < s_i$, $i = 2, 5, 6$). We simply set $s'_i = M_i$ for each $i \in \{2, 5, 6\}$, and the recursion exits. Each stratum i now has a new sample size s'_i such that $s'_i \leq s_i$ for every i, and $\sum_{i=1}^{6} s'_i = 200$.

5.3 A faster method for sample size reduction by $\beta \geq 1$ elements

We present a faster algorithm for variance-optimal sample size reduction, MultiElementSSR, with time complexity $O(r \log r)$. MultiElementSSR shares the same algorithmic foundation as SSR, but uses a faster iterative method based on sorting.

Algorithm 7: `MultiElementSSR`(M): A fast implementation of Sample Size Reduction without using recursion.

Input: The strata under consideration is $\mathcal{A} = \{1, 2, \ldots, r\}$, and the volumes and standard deviations. M is the target total sample size.
Output: For $1 \leq i \leq r$, $\mathcal{L}[i]$ is set to the final size of sample for stratum i, such that the increase of the variance V is minimized.

```
1  Allocate L[1..r], an array of numbers
2  Allocate Q[1..r], an array of (x, y, z) tuples
3  for i = 1 ... r do  Q[i] ← (i, n_i σ_i, s_i/(n_i σ_i))
4  Sort array Q in ascending order on the z dimension
5  for i = (r − 1) down to 1 do
6      Q[i].y ← Q[i].y + Q[i + 1].y
7  M_new ← M; D ← Q[1].y
8  for i = 1 ... r do
9      M_{Q[i].x} ← M · n_{Q[i].x} σ_{Q[i].x} / D
10     if s_{Q[i].x} > M_{Q[i].x} then break
11     L[Q[i].x] ← s_{Q[i].x}
12     M_new ← M_new − s_{Q[i].x}

       // Check the next sample, which must exist.
13     M_{Q[i+1].x} ← M · n_{Q[i+1].x} σ_{Q[i+1].x} / D
14     if s_{Q[i+1].x} > M_{Q[i+1].x} then                    // oversized
15         M ← M_new; D ← Q[i + 1].y

16 for j = i..r do                          // Reduce sample size to target.
17     L[Q[j].x] ← M · n_{Q[j].x} σ_{Q[j].x} / D      // Desired size for S_{Q[j].x}
18 return L
```

Definition 2 Let $Q[1..r]$ be an array of (x, y, z) tuples, where for each stratum $i = 1 \ldots r$, element $Q[i]$ is initialized as $(i, n_i\sigma_i, s_i/(n_i\sigma_i))$. Array Q is then sorted in ascending order on its z dimension.

Lemma 6 *Under memory budget M, if there exists at least one sample that is not oversized, then the collection of sample identifiers that are not oversized must occupy a continuous prefix of the array Q.*

Proof Recall that under a memory budget M, `NeyAlloc` allocates $M_i = n_i\sigma_i/D$ records to Stratum i, where $D = \sum_{i=1}^{r} n_j\sigma_j$. A sample S_i is not oversized if and only if $s_i \leq M_i$, i.e., $s_i/(n_i\sigma_i) \leq 1/D$. A sample S_i is oversized if and only if $s_i > M_i$, i.e., $s_i/(n_i\sigma_i) > 1/D$. Because array Q is in the ascending order of its z dimension, the lemma is proved. □

Lemma 6 implies that we can linearly traverse array Q from $Q[1]$ toward $Q[r]$. By comparing the sample size and the M_i for each stratum, we will be able to find the collection of samples that are not oversized, under the new target memory budget M'. After finding the prefix of the Q array that represents the collection of samples

Table 3 Time complexity of each VOR algorithm with minibatch of size b and r strata

Algorithm	Time complexity
SingleElementSSR (Algorithm 5)	$O(b)$
SSR (Algorithm 6)	$O(r^2)$
MultiElementSSR (Algorithm 7)	$O(r \log r)$

that are not oversized, we pause the walk and then set the new memory budget to be M' minus the total size of the samples in the prefix. Then, we repeat on the remaining part of the array Q (after excluding the prefix) and continue the traversal under the new memory budget. The walk will stop if we do not see any sample that is not oversized under the current memory budget M'. In that case, we can just set the size of the sample for stratum i to M'_i.

In order to avoid a re-computation of D for each new memory budget during the walk, we precompute the D for every suffix of the array Q and save the result in the y dimension of the Q array. The method MultiElementSSR in Algorithm 7 shows the pseudocode of this faster algorithm for variance-optimal sample size reduction.

Theorem 5 *(1) The* MultiElementSSR *procedure in Algorithm 7 finds the correct size of each sample of a stratified random sample, whose memory budget is reduced to M, such that the increase of the variance V is minimized. (2) The worst-case time cost of* MultiElementSSR *on a stratified random sample across r strata is* $O(r \log r)$.

Proof The correctness follows from Lemmas 4–5, Observation 1, and Lemma 6. The time complexity of MultiElementSSR is dominated by the step of sorting array Q on its z dimension (Line 4), so the worst-case time complexity of MultiElementSSR is $O(r \log r)$. □

Sample size reduction is the core to our streaming algorithms and it usually determines the time complexity for the entire algorithm. Therefore, we summarize the time complexity of each VOR algorithm in Table 3 for a concise view. Note that in both S-VOILA and SW-VOILA, we use SingleElementSSRfor evicting single element, and MultiElementSSRfor evicting multiple elements.

6 VOILA: variance-optimal offline SRS

We now present an algorithm for computing the variance-optimal allocation of sample sizes in the general case when there may be strata that are bounded. Note that once the allocation of sample sizes is determined, the actual sampling step is straightforward for the offline algorithm—samples can be chosen in a second pass through the data, using reservoir sampling within each stratum. Hence, in the rest of this section, we focus on determining the variance-optimal allocation. Consider a static data set R of n elements across r strata, where stratum i has n_i elements, and

has standard deviation σ_i. *How can a memory budget of M elements be partitioned among the strata in a variance-optimal manner?* We present VOILA (**V**ariance-**O**pt**I**ma**L** **A**llocation), an efficient offline algorithm for variance-optimal allocation that can handle strata that are bounded.

Neyman Allocation assumes there are no bounded strata (strata with small volumes). Note that it is not possible to simply eliminate strata with a low volume, by giving them full allocation, and then apply Neyman allocation on the remaining strata. The reason is as follows: suppose bounded strata are removed from further consideration. Then, remaining memory is divided among the remaining strata. This may lead to further bounded strata (which may not have been bounded earlier), and Neyman allocation again does not apply.

The following two-step process reduces variance-optimal offline SRS to variance-optimal sample size reduction.

Step 1: Suppose we start with a memory budget of n, sufficient to store all data. Then, we will just save the whole data set in the stratified random sample, and thus each sample size $s_i = n_i$. By doing so, the variance V is minimized, since $V = 0$ (Eq. 1).

Step 2: Given the stratified random sample from Step 1, we reduce the memory budget from n to M such that the resulting variance is the smallest. This can be done using variance-optimal sample size reduction, by calling SSR or MultiElementSSR with target sample size M.

VOILA (Algorithm 8) simulates this process. The algorithm only records the sample sizes of the strata in array $\mathcal{L}$, without creating the actual samples. The actual sample from stratum i is created by choosing $\mathcal{L}[i]$ elements from stratum i, using a method for uniform random sampling without replacement.

Algorithm 8: VOILA (M): Variance-optimal stratified random sampling for bounded data

```
   Input: M is the memory target
1  for i = 1...r do
2      s_i ← n_i                                   // assume total memory of n
3  L ← MultiElementSSR(M)
4  return L                  /* L[i] ≤ n_i is the sample size for stratum i in a
       variance-optimal stratified random sample . */
```

Theorem 6 *Given a data set $\mathcal{R}$ with r strata, and a memory budget M,* VOILA *(Algorithm 8) returns in $\mathcal{L}$ the sample size of each stratum in a variance-optimal stratified random sample. The worst-case time cost of* VOILA *is $O(r \log r)$.*

Proof The correctness follows from the correctness of Theorem 5, since the final sample is the sample of the smallest variance that one could obtain by reducing the initial sample (with zero variance) down to a target memory of size M. The run time is dominated by the call to MultiElementSSR, whose time complexity is $O(r \log r)$. □

7 Experimental evaluation

We present the results of an experimental evaluation. The input for our experiment is a (finite) stream of records from a data source, which is either processed by a streaming algorithm or by an offline algorithm at the end of computation. A streaming sampler must process data in a single pass using limited memory. An offline sampler has access to all data received and can compute a stratified random sample using multiple passes through data. We evaluate the samplers in two ways. The first is a direct evaluation of the sample quality through the resulting allocation and the variance of estimates obtained using the samples. The second is through the accuracy of approximate query processing using the maintained samples for different queries.

7.1 Sampling methods

We compared our stream sampling method `S-VOILA` to `Reservoir`, `ASRS`, and `Senate` sampling. `Reservoir` is a well-known stream sampling method that maintains a uniform random sample chosen without replacement from the stream—we expect the number of samples allocated to stratum i by `Reservoir` to be proportional to n_i. `Senate` [17] is a stratified sampling method that allocates each stratum an equal amount of sample space. For each stratum, `Reservoir` sampling is used to maintain a uniform sample.

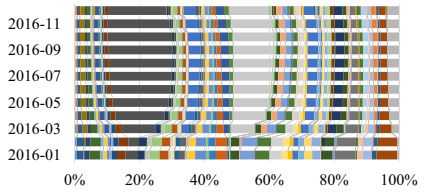
`ASRS` is an adaptive stratified sampling algorithm due to Al-kateb *et al.* (Algorithm 3 in [9]). Their algorithm considers re-allocations of memory among strata using a different method, based on power allocation [14], followed by reservoir sampling within each stratum. We chose the power allocation parameter to be 1 in order to obtain a sample of the entire population.

For streaming with sliding window, we implemented `SW-VOILA` and compared to `Reservoir` and a naive approached `S-VOILA`, which follows the original `S-VOILA` except expired elements in the sample will be removed. The statistics of strata over a sliding window could be estimated using techniques from existing work such as [41–44], which we assume available in our experiments. Note that `ASRS` does not support sliding window and thus it was excluded.

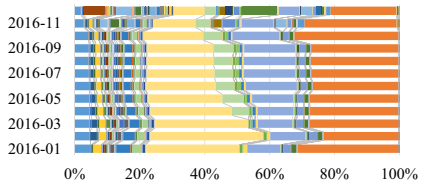
We also implemented three offline samplers `VOILA`, `NeyAlloc`, and an offline version of `Senate`. Each uses two passes to compute a stratified random sample of a total size of M records. The first pass is to determine strata characteristics used to allocate the space between strata. The second pass is to collect the samples accordingly to the computed allocation.

7.2 Data

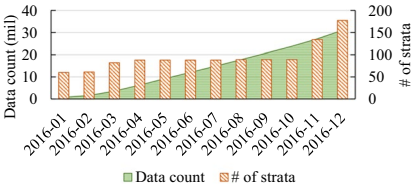
We conducted our experiments with two a real-world datasets. The first dataset called OpenAQ [45], which contains more than 31 million records of air quality measurements (concentrations of different gases and particulate matter) from



(a) Relative (cumulative) frequencies of different strata changes over time. Each color represents a stratum.



(b) Relative (cumulative) standard deviations of different strata changes over time. Each color represents a stratum



(c) The number of strata received so far, and the number of records in data.

Fig. 2 Characteristics of the OpenAQ dataset changes over time. Because the data's statistics changed significantly during the stream, the sampling algorithm has to adapt to constant changes during the stream

7, 923 locations in 62 countries around the world in 2016. Data is replayed in time order to generate the stream and is stratified based on the country of origin and the type of measurement, e.g., all measurements of carbon monoxide in the USA belong to one stratum, all records of sulfur dioxide in India belong to another stratum, and so on. The total number of strata at different points in time are shown in Fig. 2c. We also experimented with another method of stratifying data, based only on the city of origin, whose results are shown at the end of this section.

Each stratum begins with zero records, and in the initial stages, every stratum is bounded. As more data are observed, many of the strata are not bounded anymore. As Fig. 2c shows, new strata are added as more sensors are incorporated into the data stream. Figure 2a and b respectively show the cumulative frequency and standard deviation of the data over time; clearly these change significantly with time. As a result, the variance-optimal sample-size allocations to strata also change over time, and a streaming algorithm needs to adapt to these changes.

The second dataset, named Bikes, is a system logs from Divvy Bikes [46], a Chicago bike-share company. The bike-share system allows customers to pick up a bike at a station and return it to any station at their convenience. This dataset contains more than 11 million records of the bike rides collected from the year 2014 to 2018. In this paper, we stratify the data by the stations where the bikes were picked up, and we analyze the aggregation of trip duration for each stratum at the end of the month.

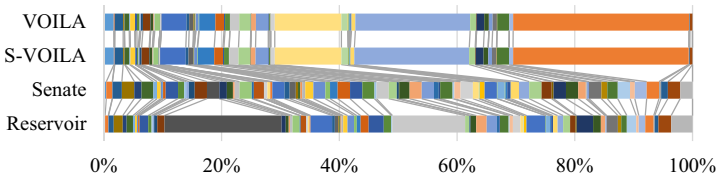


Fig. 3 Allocation of sample space among strata after 9 months of OpenAQ data. Each color represents a stratum. While `S-VOILA` has the allocation close to the offline optimal `VOILA`, `Senate` gives each stratum approximately equal amount of space. `Reservoir` space allocation matches the frequency of the data (Color figure online)

7.3 Allocations of samples to strata

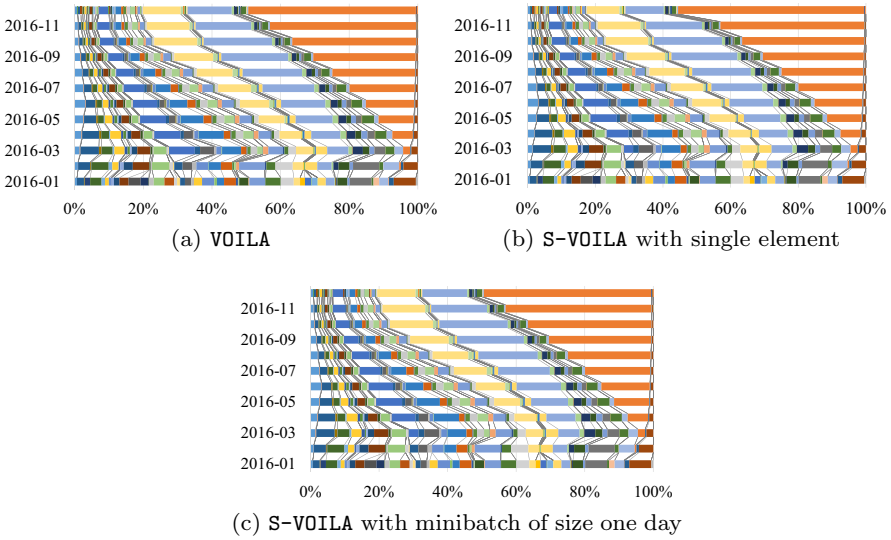
We measured the allocation of samples to different strata. Unless otherwise specified, the sample size M is set to 1 million records. For all experiments on allocations or variance, each data point is the mean of five independent runs. The allocation can be seen as a vector of numbers that sum up to M (or equivalently, normalized to sum up to 1), and we observe how this vector changes as more elements arrive.

Figure 3 shows the allocation of `Reservoir`, `Senate`, `S-VOILA` and `VOILA` algorithms at the end of September of the OpenAQ dataset. As seen, `S-VOILA` 's allocation is close to that of `VOILA`, which is optimal. Meanwhile, `Senate` allocates the memory equally among strata, despite their characteristic. `Reservoir` 's allocation fits to the frequency of the data.

Figure 4a, b and c show the change in allocations over time resulting from `VOILA`, `S-VOILA` with single element processing, and `S-VOILA` with minibatch processing. Unless otherwise specified, in the following discussion, the size of a minibatch is set to equal one day's worth of data. Visually, the allocations produced by the three methods track each other over time, showing that the streaming methods follow the allocation of `VOILA`. To understand the difference between the allocations due to `VOILA` and `S-VOILA` quantitatively, we measured the cosine distance between the allocation vectors from `VOILA` and `S-VOILA`. While detailed results are omitted due to space constraints, our results show that allocation vectors due to `S-VOILA` and `VOILA` are very similar, and the cosine distance is close to 0 most of the time and less than 0.04 at all times.

7.4 Streaming with infinite window

To evaluate the performance of each sampler over the data stream, we first assume a window with an infinite size, i.e., the elements in the data stream never expire.



(a) VOILA (b) S-VOILA with single element

(c) S-VOILA with minibatch of size one day

Fig. 4 Memory allocations of the samples of OpenAQ data change over time. The streaming algorithm S-VOILA produces sample allocations close to that of the optimal offline VOILA

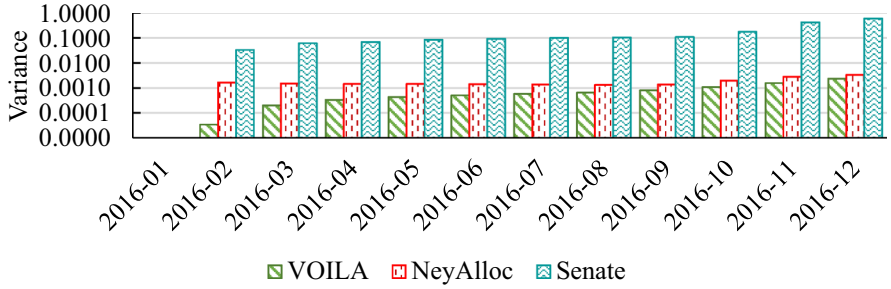


Fig. 5 The variance of VOILA compared to NeyAlloc and Senate. With the same sample size of 1M records, VOILA has a significantly smaller variance than that of NeyAlloc and Senate

7.4.1 Variance

We compared the variance of the estimates (Eq. 1) produced by different methods. The results are shown in Figs. 5 and 6. Generally, the variance of the sample due to each method increases over time, since the volume of data and the number of strata increase, while the sample size is fixed.

The comparison of different streaming algorithms is shown in Fig. 6. Among the streaming algorithms, we first note that both variants of S-VOILA have a variance that is lower than ASRS, and typically close to the optimal (VOILA). The variance of S-VOILA with minibatch processing is typically better than with single element processing. We note that the variances of both variants of S-VOILA

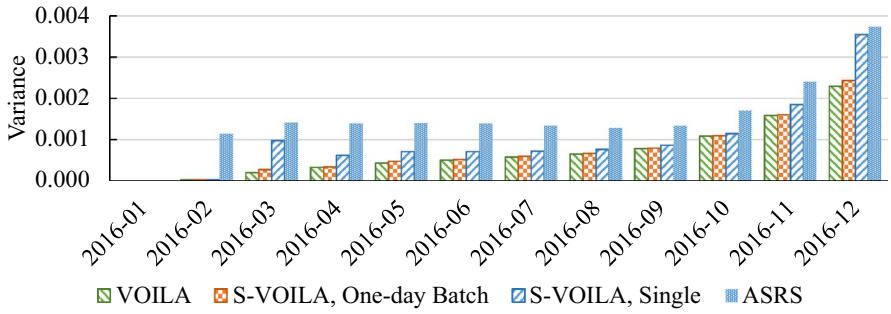


Fig. 6 The variance of streaming sampling S-VOILA compared with ASRS and the offline optimal VOILA. With the same sample size of 1M records, S-VOILA is close to VOILA. S-VOILA outperforms ASRS

are nearly equal to that of VOILA until March, when they start increasing relative to VOILA, and then converge back. From analyzing the underlying data, we see that March is the time when several new strata appear in the data (Fig. 2c), causing a substantial change in the optimal allocation of samples to strata. An offline algorithm such as VOILA can resample more elements at will since it has access to all earlier data from the stratum. However, a streaming algorithm such as S-VOILA cannot do so and must wait for enough new elements to arrive in these strata before it can "catch up" to the allocation of VOILA. Hence, S-VOILA with a single element as well as with minibatch processing show an increasing trend in the variance at such a point. When data becomes stable again the relative performance of S-VOILA improves. In November and December, new strata appear again, and the relative performance is again affected.

Among offline algorithms, we observe from Fig. 5 that Senate performs poorly since it blindly allocates equal space to all strata. NeyAlloc results in a variance that is larger than VOILA, by a factor of 1.4× to 50×. While NeyAlloc is known to be variance-optimal under the assumption of having all strata being abundant, these results show that it is far from variance-optimal for bounded str ata.

Impact of sample size To observe the sensitivity to the sample size, we conducted an experiment where the sample size is varied from 5000 to 1 million. We fixed the minibatch size to 100 thousand records. As expected, in both VOILA and S-VOILA, with single element and minibatch processing, the variance decreases when the sample size increases. The general trend was that the variance decreased by approximately a factor of 10 when the sample size increased by a factor of 10. Figure 7 shows the snapshot in September 2016 of the variance as a function of the sample size.

Impact of minibatch size We further conducted an experiment where the minibatch size is chosen from $\{1, 10, 10^2, 10^3, 10^4\}$. The results are shown in Fig. 8. A minibatch size of 10 elements yields significantly better results than single element S-VOILA. A minibatch size of 100 or greater makes the variance of S-VOILA nearly equal to the optimal variance.

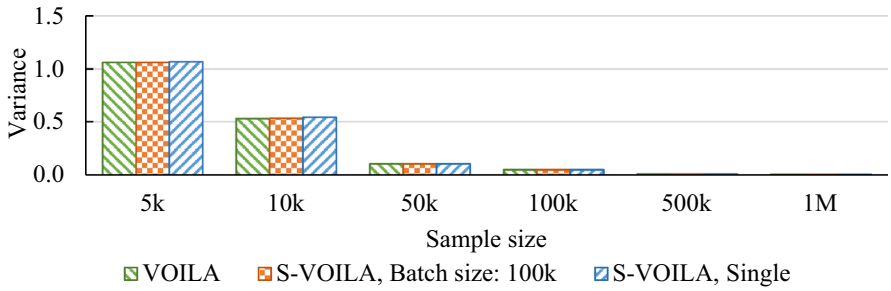


Fig. 7 Impact of Sample Size on Variance, in September, OpenAQ data

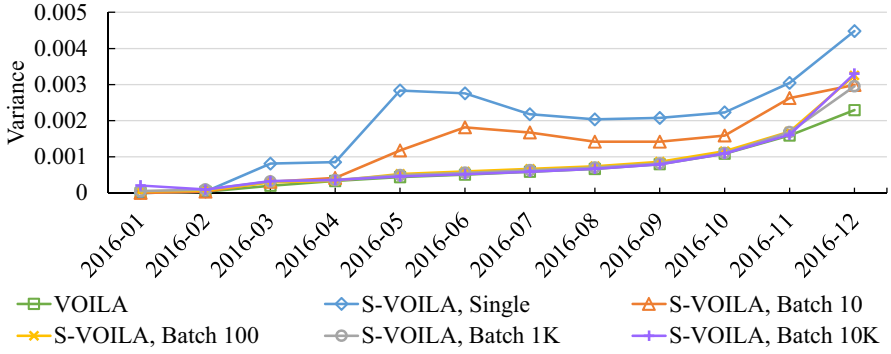


Fig. 8 Impact of Minibatch Size on Variance, OpenAQ

7.4.2 Query performance on OpenAQ

We now evaluate the quality of these samples indirectly, through their use in approximate query processing. Samples constructed using `S-VOILA` and `VOILA` are used to approximately answer a variety of queries on the data so far. For evaluating the approximation error, we also implement an exact (but expensive) method for query processing `Exact` that stores all records in a MySQL database. Identical queries are made at the same time points in the stream to the different streaming and offline samplers, as well as to the exact query processor.

A range of queries is used. Each query *selects a subset of data through a selection predicate supplied at query time, and applies an aggregate.* This shows the flexibility of the sample since it does not have any a priori knowledge of the selection predicate. We have chosen predicates with selectivity equal to one at 0.25, 0.50, and 1.00. We consider four aggregation functions: SUM, the sum of elements; SSQ, the sum of squares of elements; AVG, the mean of elements; and STD, the standard deviation. Each data point is the mean of five repetitions of the experiment with the same configuration. Each query was executed over all the

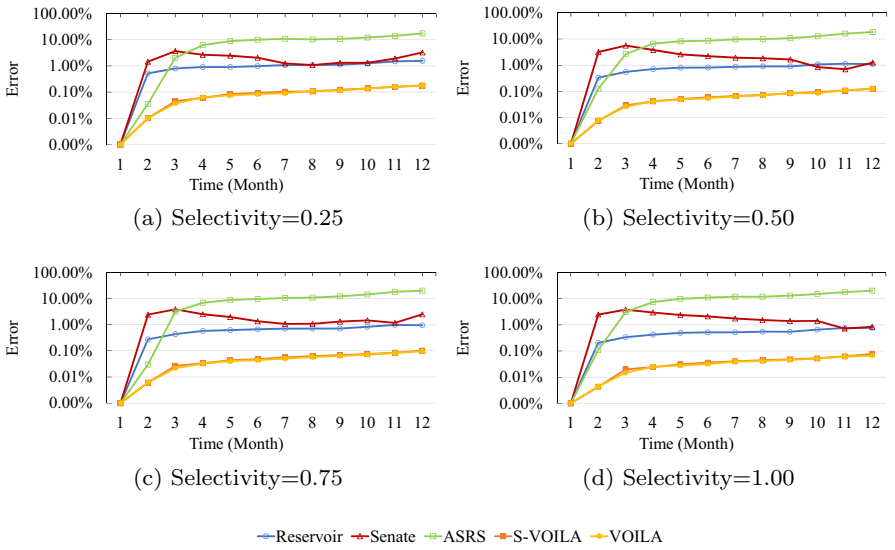


Fig. 9 Streaming samplers. SUM with different selectivity, sample size = 1 million. OpenAQ data

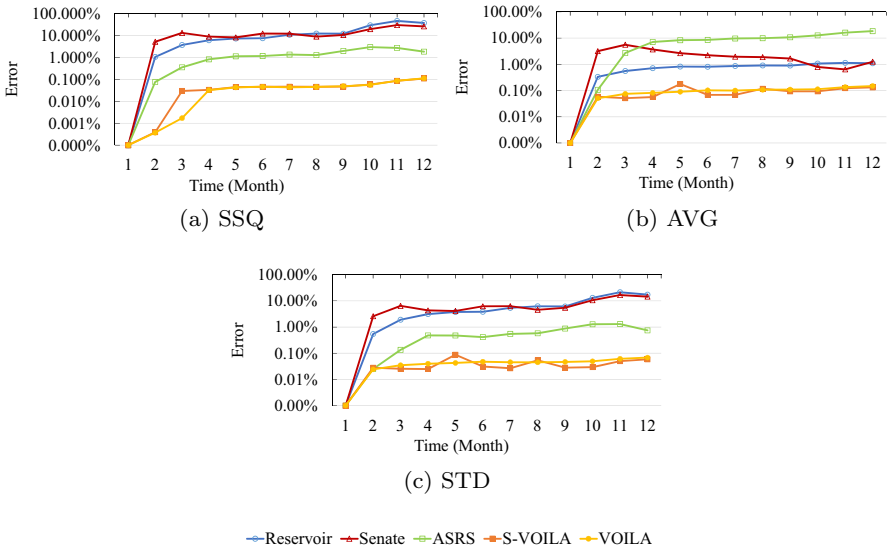


Fig. 10 Streaming samplers. SSQ, AVG, and STD with selectivity 0.50, sample size = 1 million. OpenAQ data

received data after one month of data arrived, up to the entire year of 2016 in the OpenAQ dataset with thirty-one million records.

Figures 9 and 10 shows the relative errors of different aggregations as the size of streaming data increases, while the sample size is held fixed. Both figures

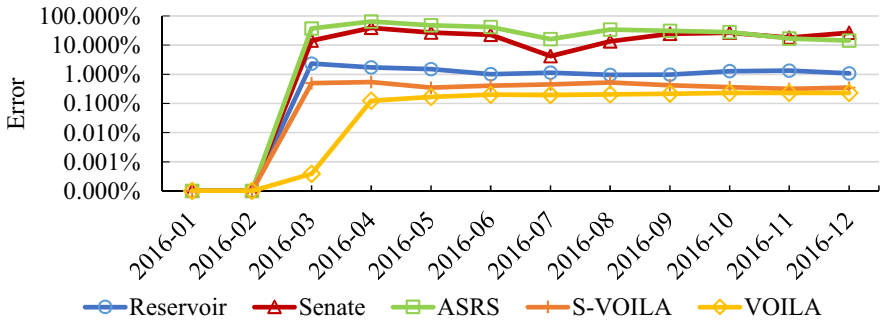


Fig. 11 Streaming samplers, data stratified by the city (SUM with selectivity 0.5)

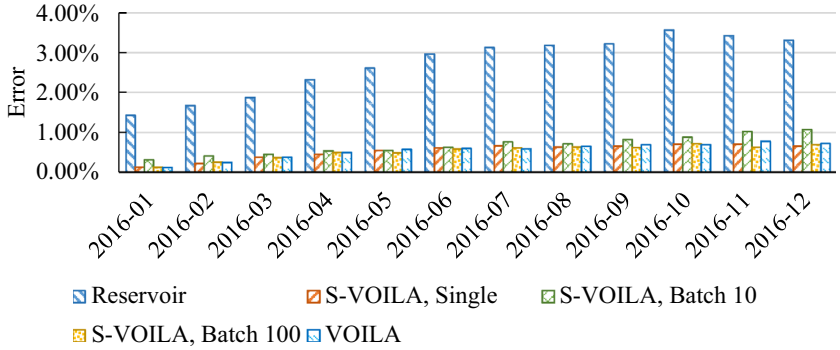
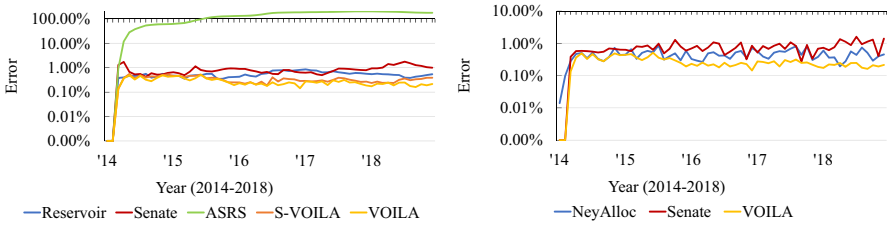


Fig. 12 Streaming samplers, impact of minibatch size, sample size = 100,000. (SUM with selectivity 0.5)

show that `S-VOILA` outperforms other streaming samplers across queries with different aggregation and selectivity. This result shows that `S-VOILA` maintains a better quality of stratified sample to answer an aggregation over a subset of data accurately. Also, `S-VOILA` performs very closely to its offline version, `VOILA`, which samples from the entire received data. We note that when `ASRS` evicts elements from per-stratum samples, there may not always be new elements to take their place, hence it often does not use its full quota of allocated memory.

Alternate methods of stratification We also experimented with the OpenAQ data set stratified differently, using the city where the observation was made. Sample results are shown in Fig. 11. We still see that `S-VOILA` outperforms `Reservoir`, `Senate`, and `ASRS`. This supports our observation that the sample maintained by `S-VOILA` is of a higher quality than other streaming samplers, no matter how data is stratified.

Impact of sample size We also explored different sample sizes varied from 500, 000 to 1 million. All methods benefit from increased sample size and the relative performance between different methods remains the same across different sizes.



(a) Streaming Samplers (b) Offline Samplers

Fig. 13 Sum query accuracy for streaming samplers with infinite window and offline samplers with sample size 100, 000. Divvy Bikes data

Impact of minibatch size Figure 12 shows the impact of the minibatch size on the accuracy of streaming samplers for the SUM query with selectivity 0.5. The sample size is set to one hundred thousand for each sampler. `S-VOILA` with different minibatch sizes has an error of less than 1%, often much smaller, while `Reservoir` has an error that is often 3% or larger. Besides, we observe that `S-VOILA` with different minibatch sizes is very close to `VOILA`.

7.4.3 Query performance on bikes

Figure 13a shows the evaluation of a SUM query on the Bikes dataset. Similar to the results on OpenAQ showed in Fig. 9, the `S-VOILA` yields the smallest error among streaming samplings, that is close to the optimal offline `VOILA`. `ASRS` performs poorly on this dataset because the number of strata in this dataset is much higher and each stratum is smaller than the ones in OpenAQ.

Figure 13b show the error for offline samplers. To no surprise, the optimal `Offline` gives the smallest error. However, the difference between errors produced by `NeyAlloc` and `VOILA` is smaller since the number of strata increased and fewer bounded strata are observed.

7.5 Streaming with sliding window

Theoretically, `SW-VOILA` could derive multiple layers of the sample in order to fully utilize the given memory budget. In practice, we observed in our experiments that `SW-VOILA` only needs two layers to fill up the memory in most cases. Thus, we use two-layer `SW-VOILA` for the following experimental studies.

We evaluate `SW-VOILA` by comparing it to `Reservoir`, `Senate`, `VOILA`, and the naive version of `S-VOILA` for the sliding window. Note that `VOILA` is an offline sampler and it serves as a reference of the optimal stratified random sample. `Reservoir`, `Senate`, and `S-VOILA` are the same streaming samplers as in the previous section, except the expired elements will now be removed from their collected samples. The removal of expired elements will prevent `Senate` and naive `S-VOILA` from fully utilizing the given sample size since they cannot easily take new elements into the sample due to the guarantee of uniformity

Fig. 14 Sample space utilization of streaming sampling algorithms `Senate`, `S-VOILA` and `SW-VOILA` changes during the stream of OpenAQ data

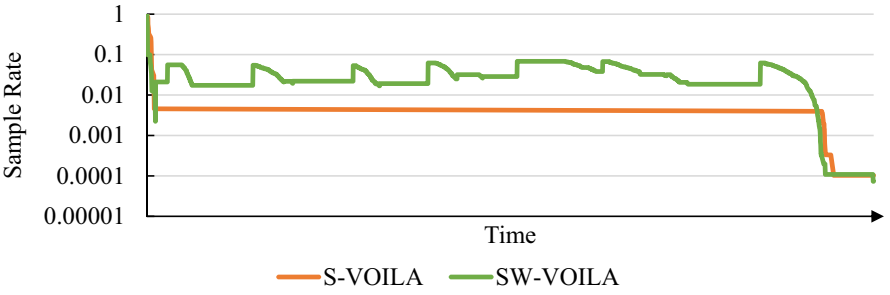


Fig. 15 Sample rate of a stratum containing measurements of NO_2 in Canada with `S-VOILA` and `SW-VOILA` on OpenAQ. While naive `S-VOILA` 's sample rate decreases, `SW-VOILA` 's sample rate increases periodically

within each stratum. For the experiment configuration, the sliding window size is set to $10E6$ and the samplers are given the maximum of $10E5$ sample each (10% sample) unless otherwise specified.

7.5.1 Memory usage

Figure 14 shows the sample size utilization of each streaming sampler at the end of each month. Each sampler is given the sample size of $10E5$, `Senate` and naive `S-VOILA` could not keep their sample full due to the expiration of old elements and the guarantee of uniformity. Most of the time, `Senate` used less than 20% and `S-VOILA` used around 80% of the given sample size, proving that to keep the uniformity within a sliding window is a challenging task. Since `Senate` has such a poor sample utilization percentage, we excluded it from the rest of our experiment comparisons since a poor performance is anticipated. On the other hand, `SW-VOILA` has used all of the given sample space throughout the entire stream. More than 97% of the given memory is used for the first (base) layer of `SW-VOILA` where uniformity of each stratum is guaranteed. The rest (3%) is

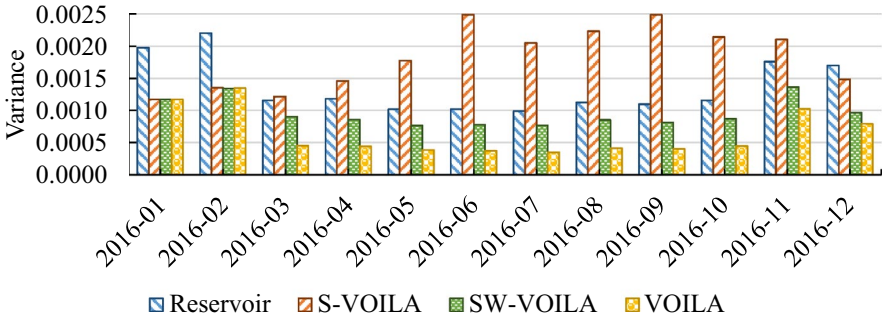


Fig. 16 Variance of `SW-VOILA` compared with `Reservoir`, `S-VOILA` and offline `VOILA` in the same window frame, OpenAQ dataset

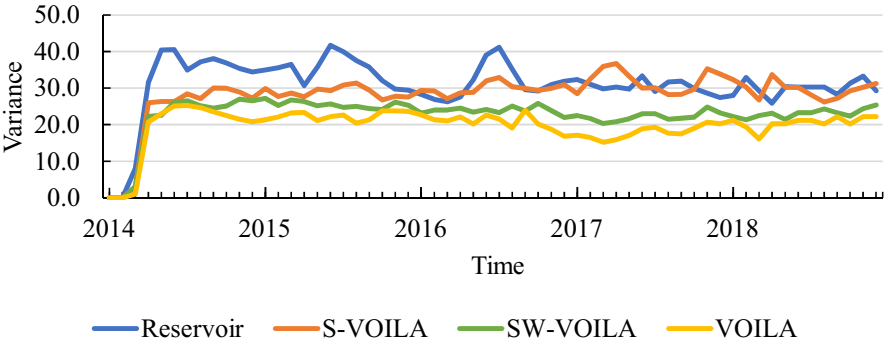


Fig. 17 Variance of `SW-VOILA` compared with `Reservoir`, `S-VOILA` and offline `VOILA` in the same window frame with sample size 100,000 and window size 1 million. Divvy Bikes data

used for the second (buffer) layer of `SW-VOILA`. Overall, the sample rates of 88 strata have increased over the stream. We see similar results on the Bikes dataset as well.

We take a closer look at `SW-VOILA` in Fig. 15, that shows the sample rate of the stratum containing measurement of NO_2 from Canada ($CA - NO_2$). Along the stream, while naive `S-VOILA` 's sample either decreased or stayed the same, the `SW-VOILA` 's sample rate periodically increased. This result shows the effect of the upper layers as the buffers for the base sample. Not only the sample size increased by merging the buffer layer and the first layer, but more importantly, the sample rate of first layer increases after merged, which means accepting new samples at a higher sample rate.

7.5.2 Variance

We measure the variance of the mean estimation, which is the objective function of our optimization problem (see Eq. 1). Figures 16 and 17 show the variance of

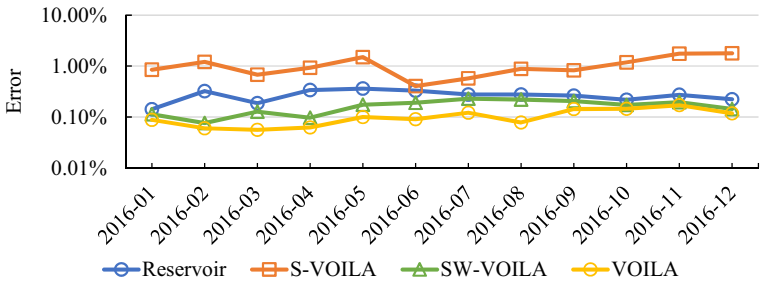


Fig. 18 Sum query accuracy of `SW-VOILA` compared with `Reservoir`, `S-VOILA` and offline `VOILA` in the same window frame, OpenAQ dataset

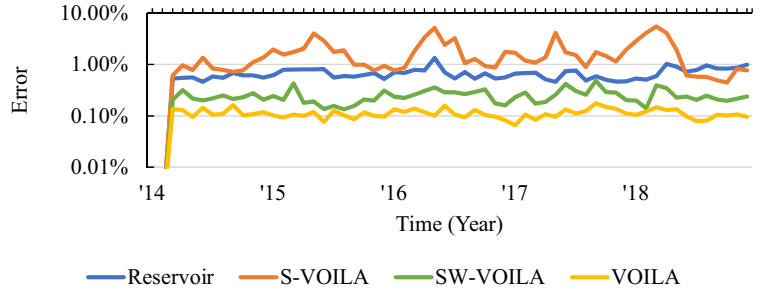


Fig. 19 Sum query accuracy of `SW-VOILA` compared with `Reservoir`, `S-VOILA` and offline `VOILA` in the same window frame with sample size 100,000 and window size 1 million. Divvy Bikes data

different samples at the end of each month for OpenAQ and Divvy Bikes datasets, respectively. In both figures, the offline optimal `VOILA` gave us a sample with the lowest variance. Among streaming samplers, `SW-VOILA` yields the smallest variance. On the OpenAQ dataset, `Reservoir` has lower variances than naive `S-VOILA` in a large portion of the stream due to unused memory space in naive `S-VOILA`. Unlike the infinite window case (Fig. 7), the variance of every sampler does not go up as the stream continues. It is because the older elements will no longer have an effect in the sample after being expired and removed.

7.5.3 Query performance

To evaluate the query performance of the sampling algorithms, we use their produced samples to answer the sum queries. As shown in Figs. 18 and 19, `SW-VOILA` yields the smallest error among all streaming samplers for both datasets. It stays close to the offline optimal `VOILA`. `S-VOILA` performs worse than `Reservoir` since the latter always utilizes all the given sample space, which also matched what we observed in Figs. 16 and 17.

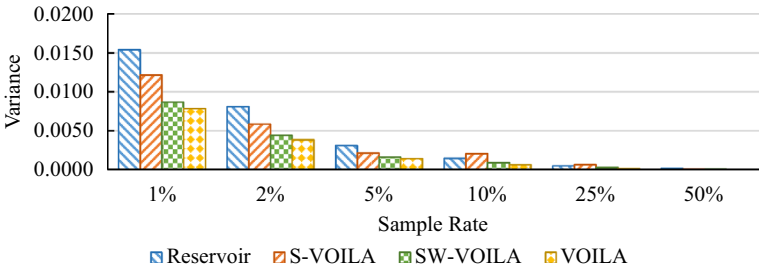


Fig. 20 Sensitivity to the sample rate: variance of the sample of sliding window size 10E6 when sample size various between 1% and 50%. OpenAQ data

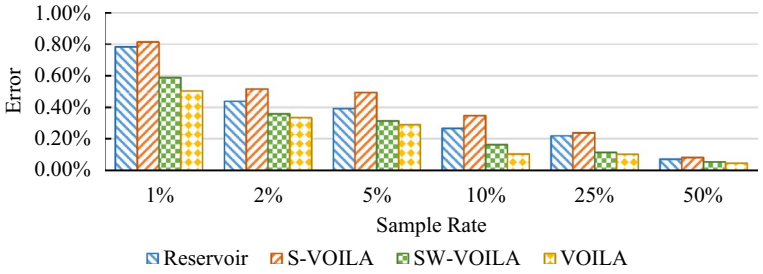


Fig. 21 Sensitivity to the sample rate: Sum query of the sample of sliding window size 10E6 when sample size various between 1% and 50%. OpenAQ data

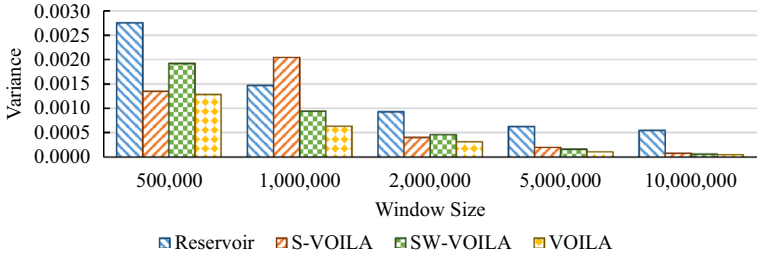


Fig. 22 Sensitivity to the window size: Variance of the sample of different window size, from 5E5 to 1E7, with 10% sample size. OpenAQ data

7.5.4 Sensitivity to the parameters

We conducted experiments to study the sensitivity of the SW-VOILA to the size of sample and the sliding window. Figures 20 and 21 show how the sample size affect the quality of the sample on variance and query performance for OpenAQ data. As expected, when the sample increases from 1 to 50%, the variances and query error of all samples decreases. Meanwhile, for each setting, VOILA always

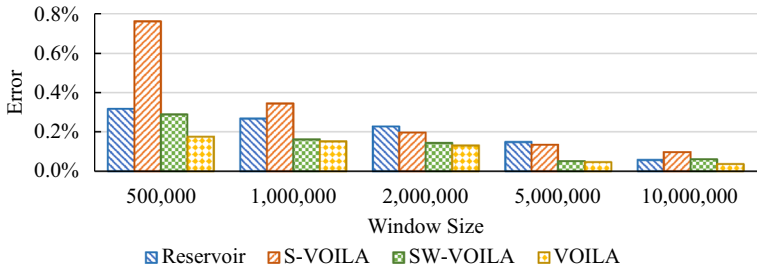


Fig. 23 Sensitivity to the window size: Sum query of the sample of different window size, from 5E5 to 1E7, with 10% sample size. OpenAQ data

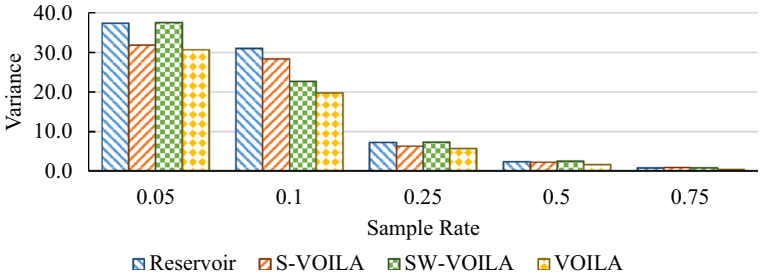


Fig. 24 Sensitivity to the sample rate: Variance of the sample of sliding window size 10E6 when sample size various between 5% and 75%. Divvy Bikes data

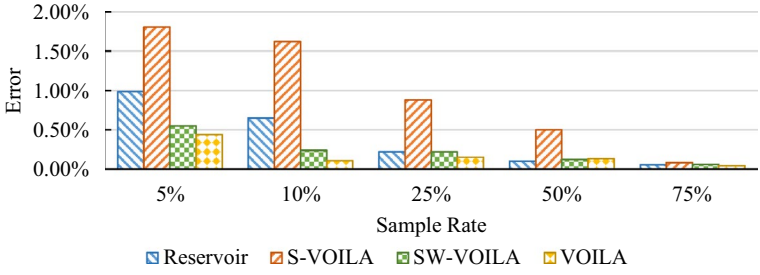


Fig. 25 Sensitivity to the sample rate: Sum query of the sample of sliding window size 10E6 when sample size various between 5% and 75%. Divvy Bikes data

provides the best sample as an offline optimal sampler. Among streaming algorithms, `SW-VOILA` provides the best sample.

Figures 22 and 23 show how the window size affects the quality of the sample for OpenAQ data when the sample size is set to 10% of the window size. As seen, the larger the window, the larger the sample size. Thus the quality of the sample increased as we increased the window size. Note that as window size increases, `S-VOILA` performs closer to `SW-VOILA` because it could be considered as a special case of `SW-VOILA` where window size is infinity.

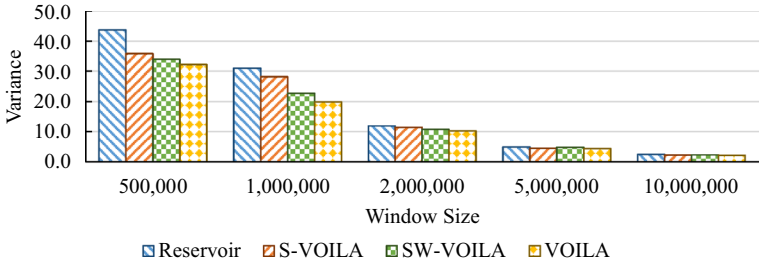


Fig. 26 Sensitivity to the window size: Variance of the sample of different window size, from 5E5 to 1E7, with 10% sample size. Divvy Bikes data

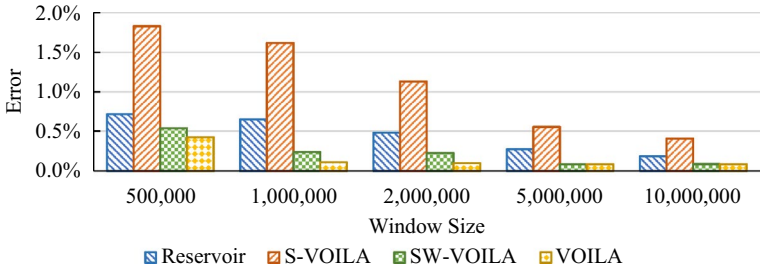


Fig. 27 Sensitivity to the window size: Sum query of the sample of different window size, from 5E5 to 1E7, with 10% sample size. Divvy Bikes data

The sensitivity tests for Divvy Bikes data were also conducted. Figures 24 and 25 show the impacts of different sample rates within the same window size. As the sample rate increases in the fixed window, the performances of `S-VOILA`, `SW-VOILA`, and the offline `VOILA` will be closer to each other. Experiments for different sliding window sizes are shown in Figs. 26 and 27. Similar results were observed comparing to OpenAQ data.

7.6 Offline sampling

We also compared `VOILA` with other offline samplers for the SUM query with different selectivities. Figure 28 shows that `VOILA` always has better performance than `Senate` and `NeyAlloc`. Our experiments with other aggregations (sum of squares, average, and standard deviation) also showed similar results.

8 Conclusions

We presented `S-VOILA`, an algorithm for streaming SRS with minibatch processing, which interleaves a continuous, locally variance-optimal re-allocation of sample sizes with streaming sampling. Our experiments show that `S-VOILA` results in variance that

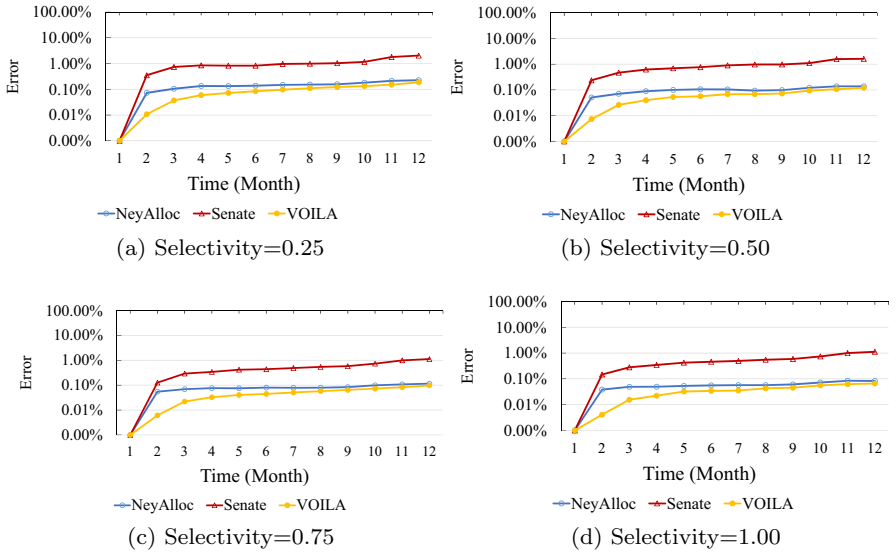


Fig. 28 Offline samplers. SUM with different selectivities, sample size = 1 million. OpenAQ data

is typically close to VOILA, which was given the entire input beforehand, and which is much smaller than that of algorithms due to prior work. We also show an inherent lower bound on the worst-case variance of any streaming algorithm for SRS—this limitation is not due to the inability to compute the optimal sample allocation in a streaming manner, but is instead due to the inability to increase sample sizes in a streaming manner, while maintaining uniformly weighted sampling within a stratum. We also investigated the more challenging case that is to maintain a streaming SRS over a sliding window. Given the high workspace lower bound we proved, we proposed a SW-VOILA, practical algorithm that can produce an SRS with good quality using a workspace that is close to the sample size. Our work also led to a variance-optimal method VOILA for offline SRS from data that may have bounded strata. Our experiments show that on real and synthetic data, an SRS obtained using VOILA can have a significantly smaller variance than one obtained by Neyman allocation.

There are several directions for future research, including (1) restratification in a streaming manner, (2) handling group-by queries and join queries, (3) incorporating general versions of time-decay, and (4) SRS on distributed data.

Acknowledgements Nguyen and Tirthapura were supported in part by NSF Grants 1527541 and 1725702.

References

1. Nguyen, T.D., Shih, M., Srivastava, D., Tirthapura, S., Xu, B.: Stratified random sampling over streaming and stored data. In: EDBT, pp. 25–36 (2019)

2. Acharya, S., Gibbons, P.B., Poosala, V., Ramaswamy, S.: The aqua approximate query answering system. In: Proceedings in SIGMOD, pp. 574–576 (1999)
3. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: BlinkDB: Queries with bounded errors and bounded response times on very large data. In: Proceedings in EuroSys, pp. 29–42 (2013)
4. Kandula, S., Shanbhag, A., Vitorovic, A., Olma, M., Grandl, R., Chaudhuri, S., Ding, B.: Quickr: lazily approximating complex adhoc queries in bigdata clusters. In: SIGMOD, pp. 631–646 (2016)
5. Chaudhuri, S., Das, G., Narasayya, V.: Optimized stratified sampling for approximate query processing. ACM TODS (2007). https://doi.org/10.1145/1242524.1242526
6. Johnson, T., Shkapenyuk, V.: Data stream warehousing in tidalrace. In: Proceeding in CIDR (2015)
7. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I.: Discretized streams: fault-tolerant streaming computation at scale. In: SOSP, pp. 423–438 (2013)
8. Neyman, J.: On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. J. R. Stat. Soc. **97**(4), 558–625 (1934)
9. Al-Kateb, M., Lee, B.S.: Adaptive stratified reservoir sampling over heterogeneous data streams. Inf. Syst. **39**, 199–216 (2014)
10. Efraimidis, P.S., Spirakis, P.G.: Weighted random sampling with a reservoir. Inf. Process. Lett. **97**(5), 181–185 (2006)
11. Meng, X.: Scalable simple random sampling and stratified sampling. In: Proceedings in ICML, pp. 531–539 (2013)
12. Al-Kateb, M., Lee, B.S.: Stratified reservoir sampling over heterogeneous data streams. In: Proceedings of SSDBM, pp. 621–639 (2010)
13. Al-Kateb, M., Lee, B.S., Wang, X.S.: Adaptive-size reservoir sampling over data streams. In: Proceedings in SSDBM, p. 22 (2007)
14. Bankier, M.D.: Power allocations: determining sample sizes for subnational areas. Am. Stat. **42**(3), 174–177 (1988)
15. Vitter, J.S.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**(1), 37–57 (1985)
16. Lang, K., Liberty, E., Shmakov, K.: Stratified sampling meets machine learning. In: Proceedings in ICML, pp. 2320–2329 (2016)
17. Acharya, S., Gibbons, P., Poosala, V.: Congressional samples for approximate answering of group-by queries. In: Proceedings in SIGMOD, pp. 487–498 (2000)
18. Babcock, B., Chaudhuri, S., Das, G.: Dynamic sample selection for approximate query processing. In: *Proceedings in SIGMOD*, pp. 539–550 (2003)
19. Joshi, S., Jermaine, C.: Robust stratified sampling plans for low selectivity queries. In: Proceedings in ICDE, pp. 199–208 (2008)
20. Ding, B., Huang, S., Chaudhuri, S., Chakrabarti, K., Wang, C.: Sample + seek: approximating aggregates with distribution precision guarantee. In: SIGMOD, pp. 679–694 (2016)
21. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceeding in PODS, pp. 1–16 (2002)
22. Cochran, W.G.: Sampling Techniques, 3rd edn. Wiley, New York (1977)
23. Haas, P.J.: Data-stream sampling: basic techniques and results. Data Stream Management, pp. 13–44. Springer, Berlin (2016)
24. Lohr, S.L.: Sampling: Design and Analysis, 2nd edn. Duxbury Press, London (2009)
25. Thompson, S.K.: Sampling, 3rd edn. Wiley, New York (2012)
26. Tillé, Y.: Sampling Algorithms, 1st edn. Springer, Berlin (2006)
27. Mcleod, I., Bellhouse, D.: A convenient algorithm for drawing a simple random sample. J. R. Stat. Soc. Ser. C **32**, 182–184 (1983)
28. Vitter, J.S.: Optimum algorithms for two random sampling problems. In: Proceeding in FOCS, pp. 65–75 (1983)
29. Braverman, V., Ostrovsky, R., Vorsanger, G.: Weighted sampling without replacement from data streams. Inf. Process. Lett. **115**(12), 923–926 (2015)
30. Gemulla, R., Lehner, W., Haas, P.J.: Maintaining bounded-size sample synopses of evolving datasets. VLDB J. **17**(2), 173–201 (2008)
31. Gibbons, P.B., Tirthapura, S.: Estimating simple functions on the union of data streams. In: *Proceedings in SPAA*, pp. 281–291 (2001)
32. Babcock, B., Datar, M., Motwani, R.: Sampling from a moving window over streaming data. In: SODA (2002)

33. Braverman, V., Ostrovsky, R., Zaniolo, C.: Optimal sampling from sliding windows. In: Proceedings in PODS, pp. 147–156 (2009)
34. Gemulla, R., Lehner, W.: Sampling time-based sliding windows in bounded space. In: SIGMOD (2008)
35. Cormode, G., Shkapenyuk, V., Srivastava, D., Xu, B.: Forward decay: a practical time decay model for streaming systems. In: Proceedings in ICDE, pp. 138–149 (2009)
36. Cormode, G., Tirthapura, S., Xu, B.: Time-decaying sketches for robust aggregation of sensor data. SIAM J. Comput. **39**(4), 1309–1339 (2009)
37. Chung, Y., Tirthapura, S.: Distinct random sampling from a distributed stream. In: IPDPS, pp. 532–541 (2015)
38. Chung, Y., Tirthapura, S., Woodruff, D.: A simple message-optimal algorithm for random sampling from a distributed stream. IEEE TKDE **28**(6), 1356–1368 (2016)
39. Cormode, G., Muthukrishnan, S., Yi, K., Zhang, Q.: Continuous sampling from distributed streams. JACM (2012). https://doi.org/10.1145/0000000.0000000
40. Tirthapura, S., Woodruff, D.P.: Optimal random sampling from distributed streams revisited. In: DISC, pp. 283–297 (2011)
41. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. SIAM J. Comput. **31**(6), 1794–1813 (2002)
42. Gibbons, P.B., Tirthapura, S.: Distributed streams algorithms for sliding windows. In: *SPAA*, pp. 63–72 (2002)
43. Babcock, B., Datar, M., Motwani, R., O'Callaghan, L.: Maintaining variance and k-medians over data stream windows. In: Proceedings of 22nd ACM Symposium on Principles of Database Systems (PODS), pp. 234–243, June (2003)
44. Zhang, L., Guan, Y.: Variance estimation over sliding windows. In: *PODS*, pp. 225–232 (2007)
45. http://openaq.org
46. https://www.divvybikes.com/system-data