

Data-Stream Sampling: Basic Techniques and Results

Peter J. Haas

1 Introduction

Perhaps the most basic synopsis of a data stream is a sample of elements from the stream. A key benefit of such a sample is its flexibility: the sample can serve as input to a wide variety of analytical procedures and can be reduced further to provide many additional data synopses. If, in particular, the sample is collected using random sampling techniques, then the sample can form a basis for statistical inference about the contents of the stream. This chapter surveys some basic sampling and inference techniques for data streams. We focus on general methods for materializing a sample; later chapters provide specialized sampling methods for specific analytic tasks.

To place the results of this chapter in context and to help orient readers having a limited background in statistics, we first give a brief overview of finite-population sampling and its relationship to database sampling. We then outline the specific data-stream sampling problems that are the subject of subsequent sections.

1.1 Finite-Population Sampling

Database sampling techniques have their roots in classical statistical methods for “finite-population sampling” (also called “survey sampling”). These latter methods are concerned with the problem of drawing inferences about a large finite *population* from a small random *sample* of population elements; see [1–5] for comprehensive

P.J. Haas (✉)
IBM Almaden Research Center, San Jose, CA, USA
e-mail: phaas@us.ibm.com

discussions. The inferences usually take the form either of testing some hypothesis about the population—e.g., that a disproportionate number of smokers in the population suffer from emphysema—or estimating some parameters of the population—e.g., total income or average height. We focus primarily on the use of sampling for estimation of population parameters.

The simplest and most common sampling and estimation schemes require that the elements in a sample be “representative” of the elements in the population. The notion of *simple random sampling* (SRS) is one way of making this concept precise. To obtain an SRS of size k from a population of size n , a sample element is selected randomly and uniformly from among the n population elements, removed from the population, and added to the sample. This sampling step is repeated until k sample elements are obtained. The key property of an SRS scheme is that each of the $\binom{n}{k}$ possible subsets of k population elements is equally likely to be produced.

Other “representative” sampling schemes besides SRS are possible. An important example is *simple random sampling with replacement* (SRSWR).¹ The SRSWR scheme is almost identical to SRS, except that each sampled element is returned to the population prior to the next random selection; thus a given population element can appear multiple times in the sample. When the sample size is very small with respect to the population size, the SRS and SRSWR schemes are almost indistinguishable, since the probability of sampling a given population element more than once is negligible. The mathematical theory of SRSWR is a bit simpler than that of SRS, so the former scheme is sometimes used as an approximation to the latter when analyzing estimation algorithms based on SRS. Other representative sampling schemes besides SRS and SRSWR include the “stratified” and “Bernoulli” schemes discussed in Sect. 2. As will become clear in the sequel, certain non-representative sampling methods are also useful in the data-stream setting.

Of equal importance to sampling methods are techniques for estimating population parameters from sample data. We discuss this topic in Sect. 4, and content ourselves here with a simple example to illustrate some of the basic issues involved. Suppose we wish to estimate the total income θ of a population of size n based on an SRS of size k , where k is much smaller than n . For this simple example, a natural estimator is obtained by scaling up the total income s of the individuals in the sample, $\hat{\theta} = (n/k)s$, e.g., if the sample comprises 1 % of the population, then scale up the total income of the sample by a factor of 100. For more complicated population parameters, such as the number of distinct ZIP codes in a population of magazine subscribers, the scale-up formula may be much less obvious. In general, the choice of estimation method is tightly coupled to the method used to obtain the underlying sample.

Even for our simple example, it is important to realize that our estimate is *random*, since it depends on the particular sample obtained. For example, suppose (rather unrealistically) that our population consists of three individuals, say Smith, Abbas, and Raman, whose respective incomes are \$10,000, \$50,000, and

¹Sometimes, to help distinguish between the two schemes more clearly, SRS is called *simple random sampling without replacement*.

Table 1 Possible scenarios, along with probabilities, for a sampling and estimation exercise

Sample	Sample income	Est. Pop. income	Scenario probability
{Smith, Abbas}	\$60,000	\$90,000	1/3
{Smith, Raman}	\$1,010,000	\$1,515,000	1/3
{Abbas, Raman}	\$1,050,000	\$1,575,000	1/3

\$1,000,000. The total income for this population is \$1,060,000. If we take an SRS of size $k = 2$ —and hence estimate the income for the population as 1.5 times the income for the sampled individuals—then the outcome of our sampling and estimation exercise would follow one of the scenarios given in Table 1. Each of the scenarios is equally likely, and the expected value (also called the “mean value”) of our estimate is computed as

$$\begin{aligned}\text{expected value} &= (1/3) \cdot (90,000) + (1/3) \cdot (1,515,000) + (1/3) \cdot (1,575,000) \\ &= 1,060,000,\end{aligned}$$

which is equal to the true answer. In general, it is important to evaluate the accuracy (degree of systematic error) and precision (degree of variability) of a sampling and estimation scheme. The *bias*, i.e., expected error, is a common measure of accuracy, and, for estimators with low bias, the *standard error* is a common measure of precision. The bias of our income estimator is 0 and the standard error is computed as the square root of the *variance* (expected squared deviation from the mean) of our estimator:

$$\begin{aligned}\text{SE} &= \left[(1/3) \cdot (90,000 - 1,060,000)^2 + (1/3) \cdot (1,515,000 - 1,060,000)^2 \right. \\ &\quad \left. + (1/3) \cdot (1,575,000 - 1,060,000)^2 \right]^{1/2} \approx 687,000.\end{aligned}$$

For more complicated population parameters and their estimators, there are often no simple formulas for gauging accuracy and precision. In these cases, one can sometimes resort to techniques based on *subsampling*, that is, taking one or more random samples from the initial population sample. Well known subsampling techniques for estimating bias and standard error include the “jackknife” and “bootstrap” methods; see [6]. In general, the accuracy and precision of a well designed sampling-based estimator should increase as the sample size increases. We discuss these issues further in Sect. 4.

1.2 Database Sampling

Although database sampling overlaps heavily with classical finite-population sampling, the former setting differs from the latter in a number of important respects.

- **Scarce versus ubiquitous data.** In the classical setting, samples are usually expensive to obtain and data is hard to come by, and so sample sizes tend to be small. In database sampling, the population size can be enormous (terabytes of data), and samples are relatively easy to collect, so that sample sizes can be relatively large [7, 8]. The emphasis in the database setting is on the sample as a flexible, lossy, compressed synopsis of the data that can be used to obtain quick approximate answers to user queries.
- **Different sampling schemes.** As a consequence of the complex storage formats and retrieval mechanisms that are characteristic of modern database systems, many sampling schemes that were unknown or of marginal interest in the classical setting are central to database sampling. For example, the classical literature pays relatively little attention to Bernoulli sampling schemes (described in Sect. 2.1 below), but such schemes are very important for database sampling because they can be easily parallelized across data partitions [9, 10]. As another example, tuples in a relational database are typically retrieved from disk in units of pages or extents. This fact strongly influences the choice of sampling and estimation schemes, and indeed has led to the introduction of several novel methods [11–13]. As a final example, estimates of the answer to an aggregation query involving select–project–join operations are often based on samples drawn individually from the input base relations [14, 15], a situation that does not arise in the classical setting.
- **No domain expertise.** In the classical setting, sampling and estimation are often carried out by an expert statistician who has prior knowledge about the population being sampled. As a result, the classical literature is rife with sampling schemes that explicitly incorporate auxiliary information about the population, as well as “model-based” schemes [4, Chap. 5] in which the population is assumed to be a sample from a hypothesized “super-population” distribution. In contrast, database systems typically must view the population (i.e., the database) as a black box, and so cannot exploit these specialized techniques.
- **Auxiliary synopses.** In contrast to a classical statistician, a database designer often has the opportunity to scan each population element as it enters the system, and therefore has the opportunity to maintain auxiliary data synopses, such as an index of “outlier” values or other data summaries, which can be used to increase the precision of sampling and estimation algorithms. If available, knowledge of the query workload can be used to guide synopsis creation; see [16–23] for examples of the use of workloads and synopses to increase precision.

Early papers on database sampling [24–29] focused on methods for obtaining samples from various kinds of data structures, as well as on the maintenance of sample views and the use of sampling to provide approximate query answers within specified time constraints. A number of authors subsequently investigated the use of sampling in query optimization, primarily in the context of estimating the size of select–join queries [22, 30–37]. Attention then shifted to the use of sampling to construct data synopses for providing quick approximate answers to decision-support queries [16–19, 21, 23]. The work in [15, 38] on online aggregation can be viewed

as a precursor to modern data-stream sampling techniques. Online-aggregation algorithms take, as input, streams of data generated by random scans of one or more (finite) relations, and produce continually-refined estimates of answers to aggregation queries over the relations, along with precision measures. The user aborts the query as soon as the running estimates are sufficiently precise; although the data stream is finite, query processing usually terminates long before the end of the stream is reached. Recent work on database sampling includes extensions of online aggregation methodology [39–42], application of bootstrapping ideas to facilitate approximate answering of very complex aggregation queries [43], and development of techniques for sampling-based discovery of correlations, functional dependencies, and other data relationships for purposes of query optimization and data integration [9, 44–46].

Collective experience has shown that sampling can be a very powerful tool, provided that it is applied judiciously. In general, sampling is well suited to very quickly identifying pervasive patterns and properties of the data when a rough approximation suffices; for example, industrial-strength sampling-enhanced query engines can speed up some common decision-support queries by orders of magnitude [10]. On the other hand, sampling is poorly suited for finding “needles in haystacks” or for producing highly precise estimates. The needle-in-haystack phenomenon appears in numerous guises. For example, precisely estimating the selectivity of a join that returns very few tuples is an extremely difficult task, since a random sample from the base relations will likely contain almost no elements of the join result [16, 31].² As another example, sampling can perform poorly when data values are highly skewed. For example, suppose we wish to estimate the average of the values in a data set that consists of 10^6 values equal to 1 and five values equal to 10^8 . The five outlier values are the needles in the haystack: if, as is likely, these values are not included in the sample, then the sampling-based estimate of the average value will be low by orders of magnitude. Even when the data is relatively well behaved, some population parameters are inherently hard to estimate from a sample. One notoriously difficult parameter is the number of distinct values in a population [47, 48]. Problems arise both when there is skew in the data-value frequencies and when there are many data values, each appearing a small number of times. In the former scenario, those values that appear few times in the database are the needles in the haystack; in the latter scenario, the sample is likely to contain no duplicate values, in which case accurate assessment of a scale-up factor is impossible. Other challenging population parameters include the minimum or maximum data value; see [49]. Researchers continue to develop new methods to deal with these problems, typically by exploiting auxiliary data synopses and workload information.

²Fortunately, for query optimization purposes it often suffices to know that a join result is “small” without knowing exactly how small.

1.3 Sampling from Data Streams

Data-stream sampling problems require the application of many ideas and techniques from traditional database sampling, but also need significant new innovations, especially to handle queries over infinite-length streams. Indeed, the unbounded nature of streaming data represents a major departure from the traditional setting. We give a brief overview of the various stream-sampling techniques considered in this chapter.

Our discussion centers around the problem of obtaining a sample from a *window*, i.e., a subinterval of the data stream, where the desired sample size is much smaller than the number of elements in the window. We draw an important distinction between a *stationary* window, whose endpoints are specified times or specified positions in the stream sequence, and a *sliding* window whose endpoints move forward as time progresses. Examples of the latter type of window include “the most recent n elements in the stream” and “elements that have arrived within the past hour.” Sampling from a finite stream is a special case of sampling from a stationary window in which the window boundaries correspond to the first and last stream elements. When dealing with a stationary window, many traditional tools and techniques for database sampling can be directly brought to bear. In general, sampling from a sliding window is a much harder problem than sampling from a stationary window: in the former case, elements must be removed from the sample as they expire, and maintaining a sample of adequate size can be difficult. We also consider “generalized” windows in which the stream consists of a sequence of transactions that insert and delete items into the window; a sliding window corresponds to the special case in which items are deleted in the same order that they are inserted.

Much attention has focused on SRS schemes because of the large body of existing theory and methods for inference from an SRS; we therefore discuss such schemes in detail. We also consider Bernoulli sampling schemes, as well as stratified schemes in which the window is divided into equal disjoint segments (the strata) and an SRS of fixed size is drawn from each stratum. As discussed in Sect. 2.3 below, stratified sampling can be advantageous when the data stream exhibits significant autocorrelation, so that elements close together in the stream tend to have similar values. The foregoing schemes fall into the category of *equal-probability sampling* because each window element is equally likely to be included in the sample. For some applications it may be desirable to bias a sample toward more recent elements. In the following sections, we discuss both equal-probability and biased sampling schemes.

2 Sampling from a Stationary Window

We consider a stationary window containing n elements e_1, e_2, \dots, e_n , enumerated in arrival order. If the endpoints of the window are defined in terms of time points t_1 and t_2 , then the number n of elements in the window is possibly random; this fact does not materially affect our discussion, provided that n is large enough so that

sampling from the window is worthwhile. We briefly discuss Bernoulli sampling schemes in which the size of the sample is random, but devote most of our attention to sampling techniques that produce a sample of a specified size.

2.1 Bernoulli Sampling

A *Bernoulli* sampling scheme with sampling rate $q \in (0, 1)$ includes each element in the sample with probability q and excludes the element with probability $1 - q$, independently of the other elements. This type of sampling is also called “binomial” sampling because the sample size is binomially distributed so that the probability that the sample contains exactly k elements is equal to $\binom{n}{k} q^k (1 - q)^{n-k}$. The expected size of the sample is nq . It follows from the central limit theorem for independent and identically distributed random variables [50, Sect. 27] that, for example, when n is reasonably large and q is not vanishingly small, the deviation from the expected size is within $\pm 100\varepsilon$ % with probability close to 98 %, where $\varepsilon = 2\sqrt{(1 - q)/nq}$. For example, if the window contains 10,000 elements and we draw a 1 % Bernoulli sample, then the true sample size will be between 80 and 120 with probability close to 98 %. Even though the size of a Bernoulli sample is random, Bernoulli sampling, like SRS and SRSWR, is a *uniform* sampling scheme, in that any two samples of the same size are equally likely to be produced.

Bernoulli sampling is appealingly easy to implement, given a pseudorandom number generator [51, Chap. 7]. A naive implementation generates for each element e_i a pseudorandom number U_i uniformly distributed on $[0, 1]$; element e_i is included in the sample if and only if $U_i \leq q$. A more efficient implementation uses the fact that the number of elements that are skipped between successive inclusions has a geometric distribution: if Δ_i is the number of elements skipped after e_i is included, then $\Pr\{\Delta_i = j\} = q(1 - q)^j$ for $j \geq 0$. To save CPU time, these random skips can be generated directly. Specifically, if U_i is a random number distributed uniformly on $[0, 1]$, then $\Delta_i = \lfloor \log U_i / \log(1 - q) \rfloor$ has the foregoing geometric distribution, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x ; see [51, p. 465]. Figure 1 displays the pseudocode for the resulting algorithm, which is executed whenever a new element e_i arrives. Lines 1–4 represent an initialization step that is executed upon the arrival of the first element (i.e., when $m = 0$ and $i = 1$). Observe that the algorithm usually does almost nothing. The “expensive” calls to the pseudorandom number generator and the $\log()$ function occur only at element-inclusion times. As mentioned previously, another key advantage of the foregoing algorithm is that it is easily parallelizable over data partitions.

A generalization of the Bernoulli sampling scheme uses a different inclusion probability for each element, including element i in the sample with probability q_i . This scheme is known as *Poisson sampling*. One motivation for Poisson sampling might be a desire to bias the sample in favor of recently arrived elements. In general, Poisson sampling is harder to implement efficiently than Bernoulli sampling because generation of the random skips is nontrivial.

```

//  $q$  is the Bernoulli sampling rate
//  $e_i$  is the element that has just arrived ( $i \geq 1$ )
//  $m$  is the index of the next element to be included (static variable initialized to 0)
//  $B$  is the Bernoulli sample of stream elements (initialized to  $\emptyset$ )
//  $\Delta$  is the size of the skip
//  $\text{random}()$  returns a uniform[0,1] pseudorandom number

1  if  $m = 0$  then                                //generate initial skip
2       $U \leftarrow \text{random}()$ 
3       $\Delta \leftarrow \lfloor \log U / \log(1 - q) \rfloor$ 
4       $m \leftarrow \Delta + 1$                         //compute index of first element to insert
5  if  $i = m$  then                                //insert element into sample and generate skip
6       $B \leftarrow B \cup \{e_i\}$ 
7       $U \leftarrow \text{random}()$ 
8       $\Delta \leftarrow \lfloor \log U / \log(1 - q) \rfloor$ 
9       $m \leftarrow m + \Delta + 1$                   //update index of next element to insert

```

Fig. 1 An algorithm for Bernoulli sampling

The main drawback of both Bernoulli and Poisson sampling is the uncontrollable variability of the sample size, which can become especially problematic when the desired sample size is small. In the remainder of this section, we focus on sampling schemes in which the final sample size is deterministic.

2.2 Reservoir Sampling

The reservoir sampling algorithm of Waterman [52, pp. 123–124] and McLeod and Bellhouse [53] produces an SRS of k elements from a window of length n , where k is specified a priori. The idea is to initialize a “reservoir” of k elements by inserting elements e_1, e_2, \dots, e_k . Then, for $i = k + 1, k + 2, \dots, n$, element e_i is inserted in the reservoir with a specified probability p_i and ignored with probability $1 - p_i$; an inserted element overwrites a “victim” that is chosen randomly and uniformly from the k elements currently in the reservoir. We denote by S_j the set of elements in the reservoir just after element e_j has been processed. By convention, we take $p_1 = p_2 = \dots = p_k = 1$. If we can choose the p_i ’s so that, for each j , the set S_j is an SRS from $U_j = \{e_1, e_2, \dots, e_j\}$, then clearly S_n will be the desired final sample. The probability that e_i is included in an SRS from U_i equals k/i , and so a plausible choice for the inclusion probabilities is given by $p_i = k/(i \vee k)$ for $1 \leq i \leq n$.³ The following theorem asserts that the resulting algorithm indeed produces an SRS.

Theorem 1 (McLeod and Bellhouse [53]) *In the reservoir sampling algorithm with $p_i = k/(i \vee k)$ for $1 \leq i \leq n$, the set S_j is a simple random sample of size $j \wedge k$ from $U_j = \{e_1, e_2, \dots, e_j\}$ for each $1 \leq j \leq n$.*

³Throughout, we denote by $x \vee y$ (resp., $x \wedge y$) the maximum (resp., minimum) of x and y .

Proof The proof is by induction on j . The assertion of the theorem is obvious for $1 \leq j \leq k$. Assume for induction that S_{j-1} is an SRS of size k from U_{j-1} , where $j \geq k + 1$. Fix a subset $A \subset U_j$ containing k elements and first suppose that $e_j \notin A$. Then

$$\begin{aligned} \Pr\{S_j = A\} &= \Pr\{S_{j-1} = A \text{ and } e_j \text{ not inserted}\} \\ &= \binom{j-1}{k}^{-1} \frac{j-k}{j} = \binom{j}{k}^{-1}, \end{aligned}$$

where the second equality follows from the induction hypothesis and the independence of the two given events. Now suppose that $e_j \in A$. For $e_r \in U_{j-1} - A$, let A_r be the set obtained from A by removing e_j and inserting e_r ; there are $j - k$ such sets. Then

$$\begin{aligned} \Pr\{S_j = A\} &= \sum_{e_r \in U_{j-1} - A} \Pr\{S_{j-1} = A_r, e_j \text{ inserted, and } e_r \text{ deleted}\} \\ &= \sum_{e_r \in U_{j-1} - A} \binom{j-1}{k}^{-1} \frac{k}{j} \frac{1}{k} = \binom{j-1}{k}^{-1} \frac{j-k}{j} = \binom{j}{k}^{-1}. \end{aligned}$$

Thus $\Pr\{S_j = A\} = 1/\binom{j}{k}$ for any subset $A \subset U_j$ of size k , and the desired result follows. \square

Efficient implementation of reservoir sampling is more complicated than that of Bernoulli sampling because of the more complicated probability distribution of the number of skips between successive inclusions. Specifically, denoting by Δ_i the number of skips before the next inclusion, given that element e_i has just been included, we have

$$f_i(m) \stackrel{\text{def}}{=} \Pr\{\Delta_i = m\} = \frac{k}{i-k} \frac{(i-k)^{\overline{m+1}}}{(i+1)^{\overline{m+1}}}$$

and

$$F_i(m) \stackrel{\text{def}}{=} \Pr\{\Delta_i \leq m\} = 1 - \frac{(i+1-k)^{\overline{m+1}}}{(i+1)^{\overline{m+1}}},$$

where $x^{\overline{n}}$ denotes the rising power $x(x+1)\cdots(x+n-1)$. Vitter [54] gives an efficient algorithm for generating samples from the above distribution. For small values of i , the fastest way to generate a skip is to use the method of *inversion*: if $F_i^{-1}(x) = \min\{m: F_i(m) \geq x\}$ and U is a random variable uniformly distributed on $[0, 1]$, then it is not hard to show that the random variable $X = F_i^{-1}(U)$ has the desired distribution function F_i , as does $X' = F_i^{-1}(1-U)$; see [51, Sect. 8.2.1]. For larger values of i , Vitter uses an *acceptance-rejection* method [51, Sect. 8.2.4]. For this method, there must exist a probability density function g_i from which it is easy

to generate sample values, along with a constant c_i —greater than 1 but as close to 1 as possible—such that $f_i(\lfloor x \rfloor) \leq c_i g_i(x)$ for all $x \geq 0$. If X is a random variable with density function g and U is a uniform random variable independent of X , then $\Pr\{\lfloor X \rfloor \leq x \mid U \leq f_i(\lfloor X \rfloor)/c_i g_i(X)\} = F_i(x)$. That is, if we generate pairs (X, U) until the relation $U \leq f_i(\lfloor X \rfloor)/c_i g_i(X)$ holds, then the final random variable X , after truncation to the nearest integer, has the desired distribution function F_i . It can be shown that, on average, c_i pairs (X, U) need to be generated to produce a sample from F_i . As a further refinement, we can reduce the number of expensive evaluations of the function f_i by finding a function h_i “close” to f_i such that h_i is inexpensive to evaluate and $h_i(x) \leq f_i(x)$ for $x \geq 0$. Then, to test whether $U \leq f_i(\lfloor X \rfloor)/c_i g_i(X)$, we first test (inexpensively) whether $U \leq h_i(\lfloor X \rfloor)/c_i g_i(X)$. Only in the rare event that this first test fails do we need to apply the expensive original test. This trick is sometimes called the “squeeze” method. Vitter shows that an appropriate choice for c_i is $c_i = (i + 1)/(i - k + 1)$, with corresponding choices

$$g_i(x) = \frac{k}{i+x} \left(\frac{i}{i+x} \right)^k \quad \text{and} \quad h_i(m) = \frac{k}{i+1} \left(\frac{i-k+1}{i+m-k+1} \right)^{k+1}.$$

Note that

$$G_i(x) = \int_0^x g_i(u) du = 1 - \left(\frac{i}{i+x} \right)^k,$$

so that, if V is a uniform random variable, then $G_i^{-1}(1 - V) = i(V^{-1/k} - 1)$ has density function g_i . Thus it is indeed easy to generate sample values from g_i .

Figure 2 displays the pseudocode for the overall algorithm; see [54] for a performance analysis and some further optimizations.⁴ As with the algorithm in Fig. 1, the algorithm in Fig. 2 is executed whenever a new element e_i arrives.

Observe that the insertion probability $p_i = k/(i \vee k)$ decreases as i increases so that it becomes increasingly difficult to insert an element into the reservoir. On the other hand, the number of opportunities for an inserted element e_i to be subsequently displaced from the sample by an arriving element also decreases as i increases. These two opposing trends precisely balance each other at all times so that the probability of being in the final sample is the same for all of the elements in the window.

Note that the reservoir sampling algorithm does not require prior knowledge of n , the size of the window—the algorithm can be terminated after any arbitrary number of elements have arrived, and the contents of the reservoir are guaranteed to be an SRS of these elements. If the window size is known in advance, then a variation of reservoir sampling, called *sequential sampling*, can be used to obtain the desired SRS of size k more efficiently. Specifically, reservoir sampling has a time complexity of $O(k + k \log(n/k))$ whereas sequential sampling has a complexity of $O(k)$. The

⁴We do not recommend the optimization given in Eq. (6.1) of [54], however, because of a potential bad interaction with the pseudorandom number generator.

```

//  $k$  is the size of the reservoir and  $n$  is the number of elements in the window
//  $e_i$  is the element that has just arrived ( $i \geq 1$ )
//  $m$  is the index of the next element  $\geq e_k$  to be included (static variable initialized to  $k$ )
//  $r$  is an array of length  $k$  containing the reservoir elements
//  $\Delta$  is the size of the skip
//  $\alpha$  is a parameter of the algorithm, typically equal to  $\approx 22k$ 
//  $\text{random}()$  returns a uniform[0,1] pseudorandom number

1  if  $i < k$  then                                     //initially fill the reservoir
2       $r[i] \leftarrow e_i$ 
3  if  $i \geq k$  and  $i = m$ 
4      //insert  $e_i$  into reservoir
5      if  $i = k$                                          //no ejection needed
6           $r[k] \leftarrow e_i$ 
7      else                                             //eject a reservoir element
8           $U \leftarrow \text{random}()$ 
9           $I \leftarrow 1 + \lfloor kU \rfloor$                      // $I$  is uniform on  $\{1, 2, \dots, k\}$ 
10          $r[I] \leftarrow e_i$ 
11     //generate the skip  $\Delta$ 
12     if  $i \leq \alpha$  then                               //use inverse transformation
13          $U \leftarrow \text{random}()$ 
14         find the smallest integer  $\Delta \geq 0$  such that
15              $(i + 1 - k)^{\Delta+1} / (i + 1)^{\Delta+1} \leq U$     //evaluate  $F_i^{-1}(1 - U)$ 
16     else
17         repeat                                         //use acceptance–rejection + squeezing
18              $V \leftarrow \text{random}()$ 
19              $X \leftarrow i(V^{-1/k} - 1)$                  //generate sample from  $g_i$  via inversion
20              $U \leftarrow \text{random}()$ 
21             if  $U \leq h_i(\lfloor X \rfloor) / c_i g_i(X)$  then break
22             until  $U \leq f_i(\lfloor X \rfloor) / c_i g_i(X)$ 
23              $\Delta \leftarrow \lfloor X \rfloor$ 
24     //update index of next element to insert
25      $m \leftarrow i + \Delta + 1$ 

```

Fig. 2 Vitter’s algorithm for reservoir sampling

sequential-sampling algorithm, due to Vitter [55], is similar in spirit to reservoir sampling, and is based on the observation that

$$\tilde{F}_{ij}(m) \stackrel{\text{def}}{=} \Pr\{\tilde{\Delta}_{ij} \leq m\} = 1 - \frac{(j-i)^{m+1}}{j^{m+1}},$$

where $\tilde{\Delta}_{ij}$ is the number of skips before the next inclusion, given that element e_{n-j} has just been included in the sample and that the sample size just after the inclusion of e_{n-j} is $|S| = k - i$. Here $x^{\underline{n}}$ denotes the falling power $x(x-1)\cdots(x-n+1)$. The sequential-sampling algorithm initially sets $i \leftarrow k$ and $j \leftarrow n$; as above, i represents the number of sample elements that remain to be selected and j represents the number of window elements that remain to be processed. The algorithm then (i) generates $\tilde{\Delta}_{ij}$, (ii) skips the next $\tilde{\Delta}_{ij}$ arriving elements, (iii) includes the next

arriving element into the sample, and (iv) sets $i \leftarrow i - 1$ and $j \leftarrow j - \tilde{\Delta}_{ij} - 1$. Steps (i)–(iv) are repeated until $i = 0$.

At each execution of Step (i), the specific method used to generate $\tilde{\Delta}_{ij}$ depends upon the current values of i and j , as well as algorithmic parameters α and β . Specifically, if $i \geq \alpha j$, then the algorithm generates $\tilde{\Delta}_{ij}$ by inversion, similarly to lines 13–15 in Fig. 2. Otherwise, the algorithm generates $\tilde{\Delta}_{ij}$ using acceptance–rejection and squeezing, exactly as in lines 17–23 in Fig. 2, but using either $c_1 = j/(j - i + 1)$,

$$g_1(x) = \begin{cases} \frac{i}{j}(1 - \frac{x}{j})^{i-1} & \text{if } 0 \leq x \leq j; \\ 0 & \text{otherwise,} \end{cases}$$

and

$$h_1(m) = \begin{cases} \frac{i}{j}(1 - \frac{m}{j-i+1})^{i-1} & \text{if } 0 \leq m \leq j - i; \\ 0 & \text{otherwise,} \end{cases}$$

or $c_2 = (i/(i - 1))((j - 1)/j)$,

$$g_2(x) = \frac{i - 1}{j - 1} \left(1 - \frac{i - 1}{j - 1}\right)^m,$$

and

$$h_2(m) = \begin{cases} \frac{i}{j}(1 - \frac{i-1}{j-m})^m & \text{if } 0 \leq m \leq j - i; \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm uses (c_1, g_1, h_1) or (c_2, g_2, h_2) according to whether $i^2/j \leq \beta$ or $i^2/j > \beta$, respectively. The values of α and β are implementation dependent; Vitter found $\alpha = 0.07$ and $\beta = 50$ optimal for his experiments, but also noted that setting $\beta \approx 1$ minimizes the average number of random numbers generated by the algorithm. See [55] for further details and optimizations.⁵

2.3 Other Sampling Schemes

We briefly mention several other sampling schemes, some of which build upon or incorporate the reservoir algorithm of Sect. 2.2.

Stratified Sampling

As mentioned before, a stratified sampling scheme divides the window into disjoint intervals, or strata, and takes a sample of specified size from each stratum. The

⁵As with the reservoir sampling algorithm in [54], we do not recommend the optimization in Sect. 5.3 of [55].

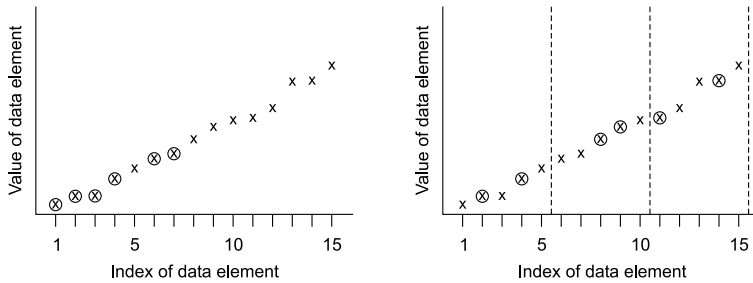


Fig. 3 (a) A realization of reservoir sampling (sample size = 6). (b) A realization of stratified sampling (sample size = 6)

simplest scheme specifies strata of approximately equal length and takes a fixed size random sample from each stratum using reservoir sampling; the random samples are of equal size.

When elements close together in the stream tend to have similar values, then the values within each stratum tend to be homogeneous so that a small sample from a stratum contains a large amount of information about all of the elements in the stratum. Figures 3(a) and 3(b) provide another way to view the potential benefit of stratified sampling. The window comprises 15 real-valued elements, and circled points correspond to sampled elements. Figure 3(a) depicts an unfortunate realization of an SRS: by sheer bad luck, the early, low-valued elements are disproportionately represented in the sample. This would lead, for example, to an underestimate of the average value of the elements in the window. Stratified sampling avoids this bad situation: a typical realization of a stratified sample (with three strata of length 5 each) might look as in Fig. 3(b). Observe that elements from all parts of the window are well represented. Such a sample would lead, e.g., to a better estimate of the average value.

Deterministic and Semi-Deterministic Schemes

Of course, the simplest scheme for producing a sample of size k inserts every m th element in the window into the sample, where $m = n/k$. There are two disadvantages to this approach. First, it is not possible to draw statistical inferences about the entire window from the sample because the necessary probabilistic context is not present. In addition, if the data in the window are periodic with a frequency that matches the sampling rate, then the sampled data will be unrepresentative of the window as a whole. For example, if there are strong weekly periodicities in the data and we sample the data every Monday, then we will have a distorted picture of the data values that appear throughout the week. One way to ameliorate the former problem is to use *systematic sampling* [1, Chap. 8]. To effect this scheme, generate a random number L between 1 and m . Then insert elements $e_L, e_{L+m}, e_{L+2m}, \dots, e_{n-m+L}$ into the sample. Statistical inference is now possible, but the periodicity issue still

remains—in the presence of periodicity, estimators based on systematic sampling can have large standard errors. On the other hand, if the data are not periodic but exhibit a strong trend, then systematic sampling can perform very well because, like stratified sampling, systematic sampling ensures that the sampled elements are spread relatively evenly throughout the window. Indeed, systematic sampling can be viewed as a type of stratified sampling where the i th stratum comprises elements $e_{(i-1)m+1}, e_{(i-1)m+2}, \dots, e_{im}$ and we sample one element from each stratum—the sampling mechanisms for the different strata are completely synchronized, however, rather than independent as in standard stratified sampling.

Biased Reservoir Sampling

Consider a generalized reservoir scheme in which the sequence of inclusion probabilities $\{p_i : 1 \leq i \leq n\}$ either is nondecreasing or does not decrease as quickly as the sequence $\{k/(i \vee k) : 1 \leq i \leq n\}$. This version of reservoir sampling favors inclusion of recently arrived elements over elements that arrived earlier in the stream.

As illustrated in Sect. 4.4 below, it can be useful to compute the marginal probability that a specified element e_i belongs to the final sample S . The probability that e_i is selected for insertion is, of course, equal to p_i . For $j > i \vee k$, the probability θ_{ij} that e_i is not displaced from the sample when element e_j arrives equals the probability that e_j is not selected for insertion plus the probability that e_j is selected but does not displace e_i . If $j \leq k$, then the processing of e_j cannot result in the removal of e_i from the reservoir. Thus

$$\theta_{ij} = (1 - p_j) + p_j \left(\frac{k-1}{k} \right) = \frac{k - p_j}{k}$$

if $j > k$, and $\theta_{ij} = 1$ otherwise. Because the random decisions made at the successive steps of the reservoir sampling scheme are mutually independent, it follows that the probability that e_i is included in S is the product of the foregoing probabilities:

$$\Pr\{e_i \in S\} = p_i \prod_{j=(i \vee k)+1}^n \frac{k - p_j}{k}. \quad (1)$$

Similar arguments lead to formulas for joint inclusion probabilities: setting $\alpha_{i,j} = \prod_{l=i}^j (k - p_l)/k$ and $\beta_{i,j} = \prod_{l=i}^j (k - 2p_l)/k$, we have, for $i < j$,

$$\Pr\{e_i, e_j \in S\} = \begin{cases} p_i \alpha_{i+1, j-1} p_j ((k-1)/k) \beta_{j+1, n} & \text{if } k \leq i < j; \\ \alpha_{k+1, j-1} p_j ((k-1)/k) \beta_{j+1, n} & \text{if } i < k < j; \\ \beta_{k+1, n} & \text{if } i < j \leq k. \end{cases} \quad (2)$$

If, for example, we set $p_i \equiv p$ for some $p \in (0, 1)$. Then, from (1),

$$\Pr\{e_i \in S\} = p \left(\frac{k-p}{k} \right)^{n-(i \vee k)}.$$

Thus the probability that element e_i is in the final sample decreases geometrically as i decreases; the larger the value of p , the faster the rate of decrease.

Chao [56] has extended the basic reservoir sampling algorithm to handle arbitrary sampling probabilities. Specifically, just after the processing of element e_i , Chao's scheme ensures that the inclusion probabilities satisfy $\Pr\{e_j \in S\} \propto r_j$ for $1 \leq j \leq i$, where $\{r_j : j \geq 1\}$ is a prespecified sequence of positive numbers. The analysis of this scheme is rather complicated, and so we refer the reader to [56] for a complete discussion.

Biased Sampling by Halving

Another way to obtain a biased sample of size k is to divide the window into L strata of $m = n/L$ elements each, denoted $\Lambda_1, \Lambda_2, \dots, \Lambda_L$, and maintain a running sample S of size k as follows. The sample is initialized as an SRS of size k from Λ_1 ; (unbiased) reservoir sampling or sequential sampling may be used for this purpose. At the j th subsequent step, $k/2$ randomly-selected elements of S are overwritten by the elements of an SRS of size $k/2$ from Λ_{j+1} (so that half of the elements in S are purged). For an element $e_i \in \Lambda_j$, we have, after the procedure has terminated,

$$\Pr\{e_i \in S\} = \frac{k}{m} \left(\frac{1}{2}\right)^{L-(j \vee 2)+1}.$$

As with biased reservoir sampling, the halving scheme ensures that the probability that e_i is in the final samples falls geometrically as i decreases. Brönnimann et al. [57] describe a related scheme when each stream element is a d -vector of 0–1 data that represents, e.g., the presence or absence in a transaction of each of d items. In this setting, the goal of each halving step is to create a subsample in which the relative occurrence frequencies of the items are as close as possible to the corresponding frequencies over all of the transactions in the original sample. The scheme uses a deterministic halving method called “epsilon approximation” to achieve this goal. The relative item frequencies in subsamples produced by this latter method tend to be closer to the relative frequencies in the original sample than are those in subsamples obtained by SRS.

3 Sampling from a Sliding Window

We now restrict attention to infinite data streams and consider methods for sampling from a sliding window that contains the most recent data elements. As mentioned previously, this task is substantially harder than sampling from a stationary window. The difficulty arises because elements must be removed from the sample as they expire so that maintaining a sample of a specified size is nontrivial. Following [58], we distinguish between *sequence-based* windows and *timestamp-based* windows. A sequence-based window of length n contains the n most recent elements,

whereas a timestamp-based window of length t contains all elements that arrived within the past t time units. Because a sliding window inherently favors recently arrived elements, we focus on techniques for equal-probability sampling from within the window itself. For completeness, we also provide a brief discussion of *generalized* windows in which elements need not leave the window in arrival order.

3.1 Sequence-Based Windows

We consider windows $\{W_j: j \geq 1\}$, each of length n , where $W_j = \{e_j, e_{j+1}, \dots, e_{j+n-1}\}$. A number of algorithms have been proposed for producing, for each window W_j , an SRS S_j of k elements from W_j . The major difference between the algorithms lies in the tradeoff between the amount of memory required and the degree of dependence between the successive S_j 's.

Complete Resampling

At one end of the spectrum, a “complete resampling” algorithm takes an independent sample from each W_j . To do this, the set of elements in the current window is buffered in memory and updated incrementally, i.e., W_{j+1} is obtained from W_j by deleting e_j and inserting e_{j+n} . Reservoir sampling (or, more efficiently, sequential sampling) can then be used to extract S_j from W_j . The S_j 's produced by this algorithm have the desirable property of being mutually independent. This algorithm is impractical, however, because it has memory and CPU requirements of $O(n)$, and n is assumed to be very large.

A Passive Algorithm

At the other end of the spectrum, the “passive” algorithm described in [58] obtains an SRS of size k from the first n elements using reservoir sampling. Thereafter, the sample is updated only when the arrival of an element coincides with the expiration of an element in the sample, in which case the expired element is removed and the new element is inserted. An argument similar to the proof of Theorem 1 shows that each S_j is a SRS from W_j . Moreover, the memory requirement is $O(k)$, the same as for the stationary-window algorithms. In contrast to complete resampling, however, the passive algorithm produces S_j 's that are highly correlated. For example, S_j and S_{j+1} are identical or almost identical for each j . Indeed, if the data elements are periodic with period n , then every S_j is identical to S_1 ; this assertion follows from the fact that if element e_i is in the sample, then so is e_{i+jn} for $j \geq 1$. Thus if S_1 is not representative, e.g., the sampled elements are clustered within W_1 as in Fig. 3(a), then each subsequent sample will suffer from the same defect.

Subsampling from a Bernoulli Sample

Babcock et al. [58] provide two algorithms intermediate to those discussed above. The first algorithm inserts elements into a set B using a Bernoulli sampling scheme; elements are removed from B when, and only when, they expire. The algorithm tries to ensure that the size of B exceeds k at all times by using an inflated Bernoulli sampling rate of $q = (2ck \log n)/n$, where c is a fixed constant. Each final sample S_j is then obtained as a simple random subsample of size k from B . An argument using Chernoff bounds (see, e.g., [59]) shows that the size of B lies between k and $4ck \log n$ with a probability that exceeds $1 - O(n^{-c})$. The S_j 's are less dependent than in the passive algorithm, but the expected memory requirement is $O(k \log n)$. Also observe that if B_j is the size of B after j elements have been processed and if $\gamma(i)$ denotes the index of the i th step at which the sample size either increases or decreases by 1, then $\Pr\{B_{\gamma(i)+1} = B_{\gamma(i)} + 1\} = \Pr\{B_{\gamma(i)+1} = B_{\gamma(i)} - 1\} = 1/2$. That is, the process $\{B_{\gamma(i)} : i \geq 0\}$ behaves like a symmetric random walk. It follows that, with probability 1, the size of the Bernoulli sample will fall below k infinitely often, which can be problematic if sampling is performed over a very long period of time.

Chain Sampling

The second algorithm, called *chain sampling*, retains the improved independence properties of the S_j 's relative to the passive algorithm, but reduces the expected memory requirement to $O(k)$. The basic algorithm maintains a sample of size 1, and a sample of size k is obtained by running k independent chain-samplers in parallel. Observe that the overall sample is therefore a simple random sample with replacement—we discuss this issue after we describe the algorithm.

To maintain a sample S of size 1, the algorithm initially inserts each newly arrived element e_i into the sample (i.e., sets the sample equal to $S = \{e_i\}$) with probability $1/i$ for $1 \leq i \leq n$. Thus the algorithm behaves initially as a reservoir sampler so that, after the n th element has been observed, S is an SRS of size 1 from $\{e_1, e_2, \dots, e_n\}$. Subsequently, whenever element e_i arrives and, just prior to arrival, the sample is $S = \{e_j\}$ with $i = j + n$ (so that the sample element e_j expires), an element randomly and uniformly selected from among $e_{j+1}, e_{j+2}, \dots, e_{j+n}$ becomes the new sample element. Observe that the algorithm does not need to store all of the elements in the window in order to replace expiring sample elements—it suffices to store a “chain” of elements associated with the sample, where the first element of the chain is the sample itself; see Fig. 4. In more detail, whenever an element e_i is added to the chain, the algorithm randomly selects the index K of the element e_K that will replace e_i upon expiration. Index K is uniformly distributed on $i + 1, i + 2, \dots, i + n$, the indexes of the elements that will be in the window just after e_i expires. When element e_K arrives, the algorithm stores e_K in memory and randomly selects the index M of the element that will replace e_K upon expiration.

To further reduce memory requirements and increase the degree of independence between successive samples, the foregoing chaining method is enhanced with a

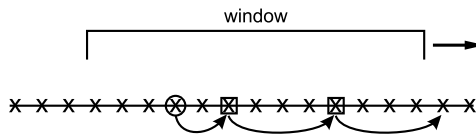


Fig. 4 Chain sampling (sample size = 1). Arrows point to the elements of the current chain, the *circled element* represents the current sample, and *elements within squares* represent those elements of the chain currently stored in memory

reservoir sampling mechanism. Specifically, suppose that element e_i arrives and, just prior to arrival, the sample is $S = \{e_j\}$ with $i < j + n$ (so that the sample element e_j does not expire). Then, with probability $1/n$, element e_i becomes the sample element; the previous sample element e_j and its associated chain are discarded, and the algorithm starts to build a new chain for the new sample element. With probability $1 - (1/n)$, element e_j remains as the sample element and its associated chain is not discarded. To see that this procedure is correct when $i < j + n$, observe that just prior to the processing of e_i , we can view S as a reservoir sample of size 1 from the “stream” of $n - 1$ elements given by $e_{i-n+1}, e_{i-n+2}, \dots, e_{i-1}$. Thus, adding e_i to the sample with probability $1/n$ amounts to executing a step of the usual reservoir algorithm, so that, after processing e_i , the set S remains an SRS of size 1 from the updated window $W_{i-n+1} = \{e_{i-n+1}, e_{i-n+2}, \dots, e_i\}$. Because the SRS property of S is preserved at each arrival epoch whether or not the current sample expires, a straightforward induction argument formally establishes that S is an SRS from the current window at all times.

Figure 5 displays the pseudocode for the foregoing algorithm; the code is executed whenever a new element e_i arrives. In the figure, the variable L denotes a linked list of chained elements of the form (e, l) , where e is an element and l is the element’s index in the stream; the list does not contain the current sample element, which is stored separately in S . Elements appear from head to tail in order of arrival, with the most recently arrived element at the tail of the list. The functions *add*, *pop*, and *purge* add a new element to the tail of the list, remove (and return the value of) the element at the head of the list, and remove all elements from the list, respectively.

We now analyze the memory requirements of the algorithm by studying the maximum amount of memory consumed during the evolution of a single chain.⁶ Denote by M the total number of elements inserted into memory during the evolution of the chain, including the initial sample. Thus $M \geq 1$ and M is an upper bound on the maximum memory actually consumed because it ignores decreases in memory consumption due to expiration of elements in the chain. Denote by X the distance from the initial sample to the next element in the chain, and recall that X is uniformly distributed on $\{1, 2, \dots, n\}$. Observe that $M \geq 2$ if and only if $X < n$ and, after the

⁶See [58] for an alternative analysis. Whenever an arriving element e_i is added to the chain and then immediately becomes the new sample element, we count this element as the first element of a new chain.

```

// n is the number of elements in the window
// ei is the element that has just arrived (i ≥ 1)
// L is a linked list (static) of chained elements (excluding sample) of the form (e, l)
// S is the sample (static, contains exactly one element)
// J is the index of the element in the sample (static, initialized to 0)
// K is the index of the next element to be added to the chain (static, initialized to 0)
// random() returns a uniform[0,1] pseudorandom number

1  if i = K                                     //add ei to chain
2      add(ei, i, L)                             //insert (ei, i) at tail of list
3      V ← random()
4      K ← i + ⌊nV⌋ + 1                         //K is uniform on i + 1, ..., i + n
5  if i = J + n                                 //current sample element is expiring
6      (e, l) ← pop(L)                             //remove element at head of list...
7      S ← {e}                                     //... to become the new sample element
8      J ← l
9  else                                         //sample element is not expiring
10     U ← random()
11     if U ≤ 1/(i ∧ n) then                     //insert ei into sample
12         S ← {ei}
13         J ← i
14     purge(L)                                   //start new chain
15     V ← random()
16     K ← i + ⌊nV⌋ + 1

```

Fig. 5 Chain-sampling algorithm (sample size = 1)

initial sample, none of the next X arriving elements become the new sample element. Thus $\Pr\{M \geq 2 \mid M \geq 1, X = j\} \leq (1 - n^{-1})^j$ for $1 \leq j \leq n$. Unconditioning on X , we have

$$\Pr\{M \geq 2 \mid M \geq 1\} \leq \sum_{j=1}^n \frac{1}{n} \left(1 - \frac{1}{n}\right)^j = 1 - \left(1 - \frac{1}{n}\right)^{n+1} \stackrel{\text{def}}{=} \beta.$$

The same argument also shows that $\Pr\{M \geq j + 1 \mid M \geq j\} \leq \beta$ for $j \geq 2$, so that $\Pr\{M \geq j\} \leq \beta^{j-1}$ for $j \geq 1$. An upper bound on the expected memory consumption is therefore given by

$$\mathbb{E}[M] = \sum_{j=1}^{\infty} \Pr\{M \geq j\} \leq \frac{1}{1 - \beta} \approx e.$$

Moreover, for $j = \alpha \ln n$ with α a fixed positive constant, we have

$$\Pr\{M \geq j + 1\} = e^{j \ln \beta} = n^{-c},$$

where $c = -\alpha \ln \beta \approx -\alpha \ln(1 - e^{-1})$. Thus the expected memory consumption for k independent samplers is $O(k)$ and, with probability $1 - O(n^{-c})$, the memory consumption does not exceed $O(k \log n)$.

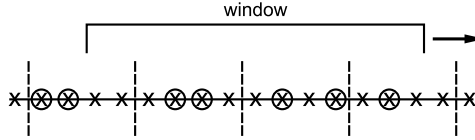


Fig. 6 Stratified sampling for a sliding window ($n = 12$, $m = 4$, $k = 2$). The *circled elements* lying within the window represent the members of the current sample, and *circled elements* lying to the left of the window represent former members of the sample that have expired

As mentioned previously, chain sampling produces an SRSWR rather than an SRS. One way of dealing with this issue is increase the size of the initial SRSWR sample S' to $|S'| = k + \alpha$, where α is large enough so that, after removal of duplicates, the size of the final SRS S will equal or exceed k with high probability. Subsampling can be then be used, if desired, to ensure that the final sample size $|S|$ equals k exactly. Using results on “occupancy distributions” [60, p. 102] it can be shown that

$$\Pr\{|S| < k\} = \sum_{j=1}^{k-1} \sum_{i=0}^j (-1)^i \binom{n}{n-j} \binom{j}{i} \left(1 - \frac{n-j+i}{n}\right)^{k+\alpha}, \quad (3)$$

and a value of α that makes the right side sufficiently small can be determined numerically, at least in principle. Assuming that $k < n/2$, a conservative but simpler approach ensures that $\Pr\{|S| < k\} < n^{-c}$ for a specified constant $c \geq 1$ by setting $\alpha = \alpha_1 \wedge \alpha_2$, where

$$\alpha_1 = \frac{(c-1) \ln n + (k+1) \ln k - (k-1) \ln 2}{\ln(n/k)}$$

and

$$\alpha_2 = c \ln n + (2ck \ln n + c^2 \ln^2 n)^{1/2}.$$

This assertion follows from a couple of simple bounding arguments.⁷

Stratified Sampling

The stratified sampling scheme for a stationary window can be adapted to obtain a stratified sample from a sliding window. The simplest scheme divides the stream into strata of length m , where m divides the window length n ; see Fig. 6. Reservoir sampling is used to obtain a SRS of size $k < m$ from each stratum. Sampled elements expire in the usual manner. The current window always contains between l and $l+1$ strata, where $l = n/m$, and all but perhaps the first and last strata are of equal length,

⁷We derive α_1 by directly bounding each term in (3). We derive α_2 by stochastically bounding $|S|$ from below by the number of successes in a sequence of $k + \alpha$ Bernoulli trials with success probability $(n - k)/n$ and then using a Chernoff bound.

namely m . The sample size fluctuates, but always lies between $k(l - 1)$ and kl . This sampling technique therefore not only retains the advantages of the stationary stratified sampling scheme but also, unlike the other sliding-window algorithms, ensures that the sample size always exceeds a specified threshold.

3.2 Timestamp-Based Windows

Relatively little is currently known about sampling from timestamp-based windows. The methods for sequence-based windows do not apply because the number of elements in the window changes over time. Babcock et al. [58] propose an algorithm called *priority sampling*. As with chain sampling, the basic algorithm maintains an SRS of size 1, and an SRSWR of size k is obtained by running k priority-samplers in parallel.

The basic algorithm for a sample size of 1 assigns to each arriving element a random priority uniformly distributed between 0 and 1. The current sample is then taken as the element in the current window having the highest priority; since each element in the window is equally likely to have the highest priority, the sample is clearly an SRS. The only elements that need to be stored in memory are those elements in the window for which there is no element with both a higher timestamp and a higher priority because only these elements can ever become the sample element. In one simple implementation, the stored elements (including the sample) are maintained as a linked list, in order of decreasing priority (and, automatically, of increasing timestamp). Each arriving element e_i is inserted into the appropriate place in the list, and all list elements having a priority smaller than that of e_i are purged, leaving e_i as the last element in the list. Elements are removed from the head of the list as they expire.

To determine the memory consumption M of the algorithm at a fixed but arbitrary time point, suppose that the window contains n elements $e_{m+1}, e_{m+2}, \dots, e_{m+n}$ for some $m \geq 0$. Denote by \mathcal{P}_i the priority of e_{m+i} , and set $\Phi_i = 1$ if e_{m+i} is currently stored in memory and $\Phi_i = 0$ otherwise. Ignore zero-probability events in which there are ties among the priorities and observe for each i that $\Phi_i = 1$ if and only if $\mathcal{P}_i > \mathcal{P}_j$ for $j = i + 1, i + 2, \dots, n$. Because priorities are assigned randomly and uniformly, each of the $n - i + 1$ elements $e_{m+i}, e_{m+i+1}, \dots, e_{m+n}$ is equally likely to be the one with the highest priority, and hence $E[\Phi_i] = \Pr\{\Phi_i = 1\} = 1/(n - i + 1)$. It follows that the expected number of elements stored in memory is

$$E[M] = E\left[\sum_{i=1}^n \Phi_i\right] = \sum_{i=1}^n E[\Phi_i] = H(n) = O(\ln n),$$

where $H(n)$ is the n th harmonic number. We can also obtain a probabilistic bound on M as follows. Denote by X_i the number of the i most recent arrivals in the window that have been inserted into the linked list: $X_i = \sum_{j=n-i+1}^n \Phi_j$. Observe

that if $X_i = m$ for some $m \geq 0$, then either $X_{i+1} = m$ or $X_{i+1} = m + 1$. Moreover, it follows from our previous analysis that $\Pr\{X_1 = 1\} = 1$ and

$$\begin{aligned} \Pr\{X_{i+1} = m_i + 1 \mid X_i = m_i, X_{i-1} = m_{i-1}, \dots, X_1 = m_1\} \\ &= \Pr\{\Phi_{n-i} = 1\} \\ &= 1/(i + 1) \end{aligned}$$

for all $1 \leq i < n$ and m_1, m_2, \dots, m_i such that $m_1 = 1$ and $m_{j+1} - m_j \in \{0, 1\}$ for $1 \leq j < i$. Thus $M = X_n$ is distributed as the number of successes in a sequence of n independent Poisson trials with success probability for the i th trial equal to $1/i$. Application of a simple Chernoff bound together with the fact that $\ln n < H(n) < 2 \ln n$ for $n \geq 3$ shows that $\Pr\{M > 2(1 + c) \ln n\} < n^{-c^2/3}$ for $c \geq 0$ and $n \geq 3$. Thus, for the overall sampling algorithm the expected memory consumption is $O(k \log n)$ and, with high probability, memory consumption does not exceed $O(k \log n)$.

3.3 Generalized Windows

In the case of both sequence-based and timestamp-based sliding windows, elements leave the window in same order that they arrive. In this section, we briefly consider a generalized setting in which elements can be deleted from a window W in arbitrary order. More precisely, we consider a set $\mathcal{T} = \{t_1, t_2, \dots\}$ of unique, distinguishable *items*, together with an infinite sequence of transactions $\gamma = (\gamma_1, \gamma_2, \dots)$. Each transaction γ_i is either of the form $+t_k$, which corresponds to the insertion of item t_k into W , or of the form $-t_k$, which corresponds to the deletion of item t_k from W . We restrict attention to sequences such that, at any time point, an item appears at most once in the window, so that the window is a true set and not a multiset. To avoid trivialities, we also require that $\gamma_n = -t_k$ only if item t_k is in the window just prior to the processing of the n th transaction. Finally, we assume throughout that the rate of insertions approximately equals the rate of deletions, so that the number of elements in the window remains roughly constant over time.

The authors in [61] provide a “random pairing” (RP) algorithm for maintaining a bounded uniform sample of W . The RP algorithm generalizes the reservoir sampling algorithm of Sect. 2.2 to handle deletions, and reduces to the passive algorithm of Sect. 3.1 when the number of elements in the window is constant over time and items are deleted in insertion order (so that W is a sequence-based sliding window).

In the RP scheme, every deletion from the window is eventually “compensated” by a subsequent insertion. At any given time, there are 0 or more “uncompensated” deletions. The RP algorithm maintains a counter c_b that records the number of “bad” uncompensated deletions in which the deleted item was also in the sample so that the sample size was decremented by 1. The RP algorithm also maintains a counter c_g that records the number of “good” uncompensated deletions in which the deleted item was not in the sample so that the sample size was not affected. Clearly, $d = c_b + c_g$ is the total number of uncompensated deletions.

```

//  $c_b$  is the # of uncompensated deletions that have been in the sample
//  $c_g$  is the # of uncompensated deletions that have not been in the sample
//  $\gamma_i$  is the transaction that has just arrived ( $i \geq 1$ )
//  $M$  is the upper bound on sample size
//  $W$  and  $S$  are the window and sample size, respectively
//  $\text{random}()$  returns a uniform[0,1] pseudorandom number

1  if  $\gamma_i = +t$  then                                     //an insertion
2      if  $c_b + c_g = 0$                                      //execute reservoir-sampling step
3          if  $|S| < M$ 
4              insert  $t$  into  $S$ 
5          else if  $\text{random}() < M/(|W| + 1)$ 
6              overwrite a randomly selected element of  $S$  with  $t$ 
7          else                                             //execute random-pairing step
8              if  $\text{random}() < c_b/(c_b + c_g)$ 
9                   $c_b \leftarrow c_b - 1$ 
10                 insert  $t$  into  $S$ 
11             else
12                  $c_g \leftarrow c_g - 1$ 
13 else                                                 //a deletion
14     if  $t \in S$ 
15          $c_b \leftarrow c_b + 1$ 
16         remove  $t$  from  $S$ 
17     else
18          $c_g \leftarrow c_g + 1$ 

```

Fig. 7 Random-pairing algorithm (simple version)

The algorithm works as follows. Deletion of an item is handled by removing the item from the sample, if present, and by incrementing the value of c_b or c_g , as appropriate. If $d = 0$, i.e., there are no uncompensated deletions, then insertions are processed as in standard RS. If $d > 0$, then we flip a coin at each insertion step, and include the incoming insertion into the sample with probability $c_b/(c_b + c_g)$; otherwise, we exclude the item from the sample. We then decrease either c_b or c_g , depending on whether the insertion has been included into the sample or not. Conceptually, whenever an item is inserted and $d > 0$, the item is paired with a randomly selected uncompensated deletion, called the “partner” deletion. The inserted item is included into the sample if its partner was in the sample at the time of its deletion, and excluded otherwise. The probability that the partner was in the sample is $c_b/(c_b + c_g)$. For purposes of sample maintenance, it is not necessary to keep track of the precise identity of the random partner; it suffices to maintain the counters c_b and c_g .

Figure 7 displays the pseudocode for the simplest version of the RP algorithm, which is executed whenever a new transaction γ_i arrives. As with classic reservoir sampling, the basic algorithm of Fig. 7 can be speeded up by directly generating random skip values; see [61] for such optimizations, as well as a correctness proof and a technique for merging samples.

Note that, if boundedness of the sample size is not a concern, then the following simple Bernoulli sampling scheme can be used to maintain S . First fix a sampling rate $q \in (0, 1)$. For an insertion transaction $\gamma_i = +t_k$, include t_k in S with probability q and exclude t_k with probability $1 - q$. For a deletion transaction $\gamma_i = -t_k$, simply remove t_k from S , if present.

In a variant of the above setting, multiple copies of each item may occur in both the window W and the sample S , so that both W and S are multisets (i.e., bags). When sampling from multisets, relatively sophisticated techniques are required to handle deletion of items. An extension of Bernoulli sampling to multisets is given in [62], and the authors in [63–65] provide techniques for maintaining a uniform sample D of the distinct items in a multiset window W and, for each item in D , an exact value (or high-precision estimate) of the frequency of the item in W .

4 Inference from a Sample

This section concerns techniques for drawing inferences about the contents of a window from a sample of window elements. As discussed in Sect. 1, such techniques belong to the domain of finite-population sampling. A complete discussion of this topic is well beyond the scope of this chapter, and so we cover only the most basic results; see [1–5] for further discussion. Our emphasis is on methods for estimating population sums and functions of such sums—these fundamental population parameters occur frequently in practice and are well understood. The “population” is, of course, the set of all elements in the window.

4.1 Estimation of Population Sums and Functions of Sums

We first describe techniques for estimating quantities of the form $\theta = \sum_{e_i \in W} h(e_i)$, where W is the window of interest, assumed to be of length n , and h is a real-valued function. We then discuss estimation methods for window characteristics of the form $\alpha = g(\theta_1, \theta_2, \dots, \theta_d)$, where $d \geq 1$. Here $g: \Re^d \mapsto \Re$ is a specified “smooth” function and each θ_j is a population sum, i.e., $\theta_j = \sum_{e_i \in W} h_j(e_i)$ for some function h_j . Some examples of these estimands are as follows, where each element e_i is a sales-transaction record and $v(e_i)$ is the dollar value of the transaction.

1. Let $h(e_i) = v(e_i)$. Then θ is the sum of sales over the transactions in the window.
2. Let h be a predicate function such that $h(e_i) = 1$ if $v(e_i) > \$1000$ and $h(e_i) = 0$ otherwise. Then θ is the total number of transactions in the window that exceed \$1000.
3. Let θ be as in example 1 above, and let $g(\theta) = \theta/n$. Then $\alpha = g(\theta)$ is the average sales amount per transaction in the window. If θ is as in example 2 above, then α is the fraction of transactions that exceed \$1000.

4. Suppose that an element not only records the dollar value of the transaction, but also enough information to compute the relative position number of the element within the window (the element in relative position 1 being the first to have arrived). Let $0 \leq w_1 \leq w_2 \leq \dots \leq w_n$ be a sequence of nondecreasing weights. If $h(e_i) = w_i v(e_i)$, then θ is a weighted sum of sales that favors more recent arrivals.
5. Let $h_1(e_i) = v(e_i)$ if $v(e_i) > \$1000$ and $h_1(e_i) = 0$ otherwise, and let h_2 be as in example 2 above. Also let θ_1 and θ_2 be the population sums that correspond to h_1 and h_2 . If $g(x, y) = x/y$, then $\alpha = g(\theta_1, \theta_2)$ is the average value of those transactions in the window that exceed \$1000.
6. Let $h_1(e_i) = v(e_i)$ and $h_2(e_i) = v^2(e_i)$. Also let $g(x, y) = (y/n) - (x/n)^2$. Then $\alpha = g(\theta_1, \theta_2)$ is the variance of the sales amounts for the transactions in the window.

As discussed in Sect. 1.1, any estimate $\hat{\theta}$ of a quantity such as θ should be supplemented with an assessment of the estimate's accuracy and precision. Typically, $\hat{\theta}$ will be (at least approximately) *unbiased*, in that $E[\hat{\theta}] = \theta$, i.e., if the sampling experiment were to be repeated multiple times, the estimator $\hat{\theta}$ would equal θ on average. In this case, the usual measure of precision is the *standard error*,⁸ defined as the standard deviation, or square root of the variance, of $\hat{\theta}$, $SE[\hat{\theta}] = E^{1/2}[(\hat{\theta} - E[\hat{\theta}])^2]$. If the distribution of $\hat{\theta}$ is approximately normal and we can compute from the sample an estimator $\hat{SE}[\hat{\theta}]$ of $SE[\hat{\theta}]$, then we can go further and make probabilistic statements of the form “with probability approximately $100p\%$, the unknown value θ lies in the (random) interval $[\hat{\theta} - z_p \hat{SE}[\hat{\theta}], \hat{\theta} + z_p \hat{SE}[\hat{\theta}]]$,” where z_p is the $(p+1)/2$ quantile of a standard (mean 0, variance 1) normal distribution. That is, we can compute *confidence intervals* for θ . A number of estimators $\hat{\theta}$, including the SRS-based expansion estimator discussed below, have approximately a normal distribution under mild regularity conditions. These conditions require, roughly speaking, that (i) the window length n be large, (ii) the sample size k be much smaller than n but reasonably large in absolute terms, say, 50 or larger, and (iii) the population sum not be significantly influenced by any one value or small group of values. Formal statements of these results take the form of “finite-population central limit theorems” [4, Sects. 3.4 and 3.5].

4.2 SRS and Bernoulli Sampling

For an SRS S of size k , the standard estimator of a population sum is the obvious one, namely the *expansion estimator* $\hat{\theta} = (n/k) \sum_{e_i \in S} h(e_i)$; cf. the example in Sect. 1.1.

⁸For a biased estimator $\hat{\theta}$, the usual precision measure is the *root mean squared error*, defined as $RMSE[\hat{\theta}] = E^{1/2}[(\hat{\theta} - \theta)^2]$. It is not hard to show that $RMSE = (\text{bias}^2 + SE^2)^{1/2}$, so that RMSE and SE coincide for an unbiased estimator.

Similarly, the estimator of the corresponding population average is the sample average $\hat{\alpha} = \hat{\theta}/n = (1/k) \sum_{e_i \in S} h(e_i)$. Both of these estimators are unbiased and *consistent*, in the sense that they converge to their true values as the sample size increases. For Bernoulli sampling with sampling rate q , the sample size k is random, and there are two possible estimators of the population sum θ : the expansion estimator (with k equal to the observed sample size) and the estimator $\theta^* = (1/q) \sum_{e_i \in S} h(e_i)$. The estimator θ^* is unbiased; the estimator $\hat{\theta}$ is slightly biased,⁹ but often has significantly lower variance than θ^* , and is usually preferred in practice. The variance of $\hat{\theta}$ is given by $\text{Var}[\hat{\theta}] = (n^2/k)(1-f)\sigma^2$ under SRS (exactly) and under Bernoulli sampling (approximately), where $f = k/n$ is the sampling fraction and

$$\sigma^2 = \frac{\sum_{e_i \in W} (h(e_i) - (\theta/n))^2}{n-1} \quad (4)$$

is the variance¹⁰ of the numbers $\{h(e_i) : e_i \in W\}$. An unbiased estimator of $\text{Var}[\hat{\theta}]$, based on the values in the sample, is $\hat{\text{Var}}[\hat{\theta}] = (n^2/k)(1-f)\hat{\sigma}^2$, where $\hat{\sigma}^2 = (1/(k-1)) \sum_{e_i \in S} (h(e_i) - (\hat{\theta}/n))^2$. To obtain corresponding variance formulas for the sample average α , simply multiply by n^{-2} , i.e., $\text{Var}[\hat{\alpha}] = (\sigma^2/k)(1-f)$ and $\hat{\text{Var}}[\hat{\alpha}] = (\hat{\sigma}^2/k)(1-f)$. To obtain expressions for standard errors and their estimators, take the square root of the corresponding expressions for variances.

4.3 Stratified Sampling

Here we consider the simple stratified sampling scheme for a stationary window discussed in Sect. 2.3. Our results also apply to the stratified sampling scheme for sliding windows in Sect. 3.1, at those times when the window boundaries align with the strata boundaries; the modifications required to deal with arbitrary time points can be derived, e.g., from the results in [1, Chap. 5]. Suppose that our goal is to estimate an unknown population sum θ . Also suppose that there are L equal-sized strata, with $m = n/L$ elements per stratum, and that we obtain an SRS of $r = k/L$ elements from each stratum. Denote by Λ_j the set of elements in the j th stratum and by S_j the elements of Λ_j that are in the final sample. The usual expansion estimator $\hat{\theta}$ is unbiased for θ , and

$$\text{Var}[\hat{\theta}] = m \left(\frac{n}{k} - 1 \right) \sum_{j=1}^L \sigma_j^2,$$

⁹The bias arises from the fact that the sample can be empty, albeit typically with low probability.

¹⁰We follow the survey-sampling literature, which usually takes the denominator in (4) as $n-1$ instead of n because this convention leads to simpler formulas.

where $\sigma_j^2 = (1/(m-1)) \sum_{e_i \in \Lambda_j} (h(e_i) - (\theta_j/m))^2$ and $\theta_j = \sum_{e_i \in \Lambda_j} h(e_i)$. An unbiased estimator of $\text{Var}[\hat{\theta}]$ is

$$\hat{\text{Var}}[\hat{\theta}] = m \left(\frac{n}{k} - 1 \right) \sum_{j=1}^L \hat{\sigma}_j^2,$$

where $\hat{\sigma}_j^2 = (1/(r-1)) \sum_{e_i \in S_j} (h(e_i) - (\hat{\theta}_j/m))^2$ and $\hat{\theta}_j = (m/r) \sum_{e_i \in S_j} h(e_i)$.

Observe that if the strata are highly homogeneous, then each σ_j^2 is very small, so that $\text{Var}[\hat{\theta}]$ is very small. Indeed, it can be shown [1, Sect. 5.6] that if $m \sum_{j=1}^L ((\theta_j/m) - (\theta/n))^2 \geq (1 - (m/n)) \sum_{j=1}^L \sigma_j^2$, then the variance of $\hat{\theta}$ under stratified sampling is less than or equal to the variance under simple random sampling. This condition holds except when the stratum means are almost equal, i.e., if the data are even slightly stratified, then stratified sampling yields more precise results than SRS.

4.4 Biased Sampling

When using a biased sampling scheme, we can, in principle, recover an unbiased estimate of a population sum by using a *Horvitz–Thompson* (HT) estimator; see, for example, [3], where these estimators are called π -estimators. The general form of an HT-estimator for a population sum of the form $\theta = \sum_{i \in W} h(e_i)$ based on a sample $S \subseteq W$ is $\hat{\theta}_{\text{HT}} = \sum_{i \in S} (h(e_i)/\pi_i)$, where π_i is the probability that element e_i is included in S . Assume that $\pi_i > 0$ for each e_i , and let $\Phi_i = 1$ if $e_i \in S$ and $\Phi_i = 0$ otherwise, so that $E[\Phi_i] = \pi_i$. Observe that

$$E[\hat{\theta}] = E \left[\sum_{i \in W} h(e_i) \Phi_i / \pi_i \right] = \sum_{i \in W} h(e_i) E[\Phi_i] / \pi_i = \theta, \quad (5)$$

so that HT-estimators are indeed unbiased. Similar calculations [3, Result 2.8.1] show that the variance of $\hat{\theta}$ is given by

$$\text{Var}[\hat{\theta}] = \sum_{i \in W} \sum_{j \in W} \left(\frac{\pi_{ij}}{\pi_i \pi_j} - 1 \right) h(e_i) h(e_j),$$

where π_{ij} is the probability that elements e_i and e_j are both included in S . (Take $\pi_{ii} = \pi_i$ for $i \in W$.) When using biased reservoir sampling, for example, the probabilities π_i and π_{ij} can be determined from (1) and (2). (In this case, and in others arising in practice, it can be expensive to compute the π_i 's and π_{ij} 's.) Provided that $\pi_{ij} > 0$ for all i, j , an unbiased estimator of $\text{Var}[\hat{\theta}]$ is given by

$$\hat{\text{Var}}[\hat{\theta}] = \sum_{i \in S} \sum_{j \in S} \left(\frac{1}{\pi_i \pi_j} - \frac{1}{\pi_{ij}} \right) h(e_i) h(e_j).$$

It can be shown [3, Result 2.8.2] that if the sampling scheme is such that the final sample size is deterministic and each π_{ij} is positive, then alternative forms for the variance and variance estimator are given by

$$\text{Var}[\hat{\theta}] = \frac{1}{2} \sum_{i \in W} \sum_{j \in W} (\pi_i \pi_j - \pi_{ij}) \left(\frac{h(e_i)}{\pi_i} - \frac{h(e_j)}{\pi_j} \right)^2$$

and

$$\hat{\text{Var}}[\hat{\theta}] = \frac{1}{2} \sum_{i \in S} \sum_{j \in S} \left(\frac{\pi_i \pi_j}{\pi_{ij}} - 1 \right) \left(\frac{h(e_i)}{\pi_i} - \frac{h(e_j)}{\pi_j} \right)^2.$$

The latter variance estimator is known as the *Yates–Grundy–Sen* estimator; unlike the previous variance estimator, it has the advantage of always being nonnegative if each term $(\pi_i \pi_j / \pi_{ij}) - 1$ is positive (but is only guaranteed to be unbiased for fixed-size sampling schemes). Most of the estimators discussed previously can be viewed as HT-estimators, for example, the SRS-based expansion estimator: from (1) and (2), we have $\pi_i = k/n$ and $\pi_{ij} = k(k-1)/(n(n-1))$ for all i, j .

In general, quantifying the effects of biased sampling on the outcome of a subsequent data analysis can be difficult. When estimating a population sum, however, the foregoing results lead to a clear understanding of the consequences of biased sampling schemes. Specifically, observe that, by (5), the sample sum $\hat{\theta} = \sum_{e_i \in S} h(e_i)$ is, in fact, an unbiased estimator of the *weighted* sum $\theta = \sum_{e_i \in W} \pi_i h(e_i)$.

4.5 Functions of Population Sums

Suppose that we wish to estimate $\alpha = g(\theta)$, where $\theta = (\theta_1, \theta_2, \dots, \theta_d)$ is a vector of population sums corresponding to real-valued functions h_1, h_2, \dots, h_d . We assume that there exists an unbiased and consistent estimator $\hat{\theta}_j$ for each θ_j , and we write $\hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_d)$. We also assume that g is continuous and differentiable in a neighborhood of θ and write $\nabla g = (\nabla_1 g, \nabla_2 g, \dots, \nabla_d g)$ for the gradient of g . A straightforward estimate of α is $\hat{\alpha} = g(\hat{\theta})$, i.e., we simply replace each population sum θ_j by its estimate and then apply the function g . The estimator $\hat{\alpha}$ will in general be biased if g is a nonlinear function. For example, Jensen's inequality [50, Sect. 21] implies that $E[\hat{\alpha}] \geq \alpha$ if g is convex and $E[\hat{\alpha}] \leq \alpha$ if g is concave. The bias decreases, however, as the sample size increases and, moreover, $\hat{\alpha}$ is consistent for α since g is continuous. The variance of $\hat{\alpha}$ is difficult to obtain precisely. If the sample size is large, however, so that with high probability $\hat{\theta}$ is close θ , then we can approximate $\text{Var}[\hat{\alpha}]$ by the variance of a linearized version of α obtained by taking a Taylor expansion around the point θ . That is,

$$\text{Var}[\hat{\alpha}] \approx \text{Var} \left[g(\theta) + \sum_{j=1}^d a_j (\hat{\theta}_j - \theta_j) \right] = \sum_{i=1}^d \sum_{j=1}^d a_i a_j \text{Cov}[\hat{\theta}_i, \hat{\theta}_j],$$

where $a_i = \nabla_i g(\theta)$ and $\text{Cov}[\hat{\theta}_i, \hat{\theta}_j]$ denotes the covariance of $\hat{\theta}_i$ and $\hat{\theta}_j$. We estimate $\text{Var}[\hat{\alpha}]$ by $\hat{\text{Var}}[\hat{\alpha}] = \sum_{i=1}^d \sum_{j=1}^d \hat{a}_i \hat{a}_j \hat{\text{Cov}}[\hat{\theta}_i, \hat{\theta}_j]$, where $\hat{a}_i = \nabla_i g(\hat{\theta})$ and $\hat{\text{Cov}}[\hat{\theta}_i, \hat{\theta}_j]$ is an estimate of $\text{Cov}[\hat{\theta}_i, \hat{\theta}_j]$. The exact formula for the covariance estimator depends on the specific sampling scheme and population-sum estimator. Typically, assuming that $\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_d$ are computed from the same sample, such formulas are directly analogous to those for the variance. For example, for the expansion estimator under SRS, we have

$$\hat{\text{Cov}}[\hat{\theta}_i, \hat{\theta}_j] = \frac{n^2(1-f)}{k} \frac{1}{k-1} \sum_{e_l \in S} (h_i(e_l) - (\hat{\theta}_i/n))(h_j(e_l) - (\hat{\theta}_j/n)),$$

where $f = k/n$ and $\hat{\theta}_j = (n/k) \sum_{e_l \in S} h_j(e_l)$ for each j . For stratified sampling with L strata of length m ,

$$\hat{\text{Cov}}[\hat{\theta}_i, \hat{\theta}_j] = m \left(\frac{n}{k} - 1 \right) \sum_{q=1}^L \frac{1}{r-1} \sum_{e_l \in S_q} (h_i(e_l) - (\hat{\theta}_{i,q}/m))(h_j(e_l) - (\hat{\theta}_{j,q}/m)),$$

where, as before, each stratum comprises $m = n/L$ elements, S_q is the SRS of size $r = k/L$ from the q th stratum, and $\hat{\theta}_{s,q} = (m/r) \sum_{e_l \in S_q} h_s(e_l)$ for each s and q . In general, we note that for any sampling scheme (possibly biased) such that the size of the sample S is deterministic and each π_{ij} is positive, the linearization approach leads to the following Yates–Grundy–Sen variance estimator:

$$\hat{\text{Var}}[\hat{\alpha}] = \frac{1}{2} \sum_{i \in S} \sum_{j \in S} \left(\frac{\pi_i \pi_j}{\pi_{ij}} - 1 \right) \left(\frac{z_i}{\pi_i} - \frac{z_j}{\pi_j} \right)^2,$$

where $z_l = \sum_{j=1}^L \nabla_j g(\hat{\theta}) h_j(e_l)$ for each $e_l \in S$; see [4, Sect. 4.2.1].

Acknowledgements The author would like to thank P. Brown, M. Datar, and Rainer Gemulla for several helpful discussions. D. Sivakumar suggested the idea underlying the bound α_2 given in Sect. 3.1.

References

1. W.G. Cochran, *Sampling Techniques*, 3rd edn. (Wiley, New York, 1977)
2. L. Kish, *Survey Sampling* (Wiley, New York, 1965)
3. C.E. Särndal, B. Swensson, J. Wretman, *Model Assisted Survey Sampling* (Springer, New York, 1992)
4. M.E. Thompson, *Theory of Sample Surveys* (Chapman & Hall, London, 1997)
5. S.K. Thompson, *Sampling* (Wiley, New York, 2002)
6. B. Efron, R.J. Tibshirani, *An Introduction to the Bootstrap* (Chapman & Hall, New York, 1993)
7. R. Gemulla, W. Lehner, Deferred maintenance of disk-based random samples, in *Proc. EDBT*. Lecture Notes in Computer Science (Springer, Berlin, 2006), pp. 423–441

8. A. Pol, C.M. Jermaine, S. Arumugam, Maintaining very large random samples using the geometric file. *VLDB J.* **17**(5), 997–1018 (2008)
9. P.G. Brown, P.J. Haas, Techniques for warehousing of sample data, in *Proc. 22nd ICDE* (2006)
10. P.J. Haas, The need for speed: speeding up DB2 using sampling. *IDUG Solut. J.* **10**, 32–34 (2003)
11. S. Chaudhuri, R. Motwani, V.R. Narasayya, Random sampling for histogram construction: how much is enough? in *Proc. ACM SIGMOD* (1998), pp. 436–447
12. D. DeWitt, J.F. Naughton, D.A. Schneider, S. Seshadri, Practical skew handling algorithms for parallel joins, in *Proc. 19th VLDB* (1992), pp. 27–40
13. P.J. Haas, C. König, A bi-level Bernoulli scheme for database sampling, in *Proc. ACM SIGMOD* (2004), pp. 275–286
14. W. Hou, G. Ozsoyoglu, B. Taneja, Statistical estimators for relational algebra expressions, in *Proc. Seventh PODS* (1988), pp. 276–287
15. P.J. Haas, J.M. Hellerstein, Ripple joins for online aggregation, in *Proc. ACM SIGMOD* (1999), pp. 287–298
16. S. Acharya, P. Gibbons, V. Poosala, S. Ramaswamy, Join synopses for approximate query answering, in *Proc. ACM SIGMOD* (1999), pp. 275–286
17. S. Acharya, P. Gibbons, V. Poosala, Congressional samples for approximate answering of group-by queries, in *Proc. ACM SIGMOD* (2000), pp. 487–498
18. S. Chaudhuri, G. Das, M. Datar, R. Motwani, V.R. Narasayya, Overcoming limitations of sampling for aggregation queries, in *Proc. Seventeenth ICDE* (2001), pp. 534–542
19. S. Chaudhuri, R. Motwani, V.R. Narasayya, On random sampling over joins, in *Proc. ACM SIGMOD* (1999), pp. 263–274
20. S. Ganguly, P.B. Gibbons, Y. Matias, A. Silberschatz, Bifocal sampling for skew-resistant join size estimation, in *Proc. ACM SIGMOD* (1996), pp. 271–281
21. V. Ganti, M.L. Lee, R. Ramakrishnan, ICICLES: self-tuning samples for approximate query answering, in *Proc. 26th VLDB* (2000), pp. 176–187
22. P.J. Haas, A.N. Swami, Sampling-based selectivity estimation using augmented frequent value statistics, in *Proc. Eleventh ICDE* (1995), pp. 522–531
23. C. Jermaine, Robust estimation with sampling and approximate pre-aggregation, in *Proc. 29th VLDB* (2003), pp. 886–897
24. W. Hou, G. Ozsoyoglu, B. Taneja, Processing aggregate relational queries with hard time constraints, in *Proc. ACM SIGMOD* (1989), pp. 68–77
25. F. Olken, D. Rotem, Simple random sampling from relational databases, in *Proc. 12th VLDB* (1986), pp. 160–169
26. F. Olken, D. Rotem, Random sampling from B^+ trees, in *Proc. 15th VLDB* (1989), pp. 269–277
27. F. Olken, D. Rotem, Maintenance of materialized views of sampling queries, in *Proc. Eighth ICDE* (1992), pp. 632–641
28. F. Olken, D. Rotem, Sampling from spatial databases, in *Proc. Ninth ICDE* (1993), pp. 199–208
29. F. Olken, D. Rotem, P. Xu, Random sampling from hash files, in *Proc. ACM SIGMOD* (1990), pp. 375–386
30. P.J. Haas, J.F. Naughton, S. Seshadri, A.N. Swami, Selectivity and cost estimation for joins based on random sampling. *J. Comput. Syst. Sci.* **52**, 550–569 (1996)
31. P.J. Haas, J.F. Naughton, A.N. Swami, On the relative cost of sampling for join selectivity estimation, in *Proc. Thirteenth PODS* (1994), pp. 14–24
32. P.J. Haas, A.N. Swami, Sequential sampling procedures for query size estimation, in *Proc. ACM SIGMOD* (1992), pp. 1–11
33. W. Hou, G. Ozsoyoglu, E. Dogdu, Error-constrained COUNT query evaluation in relational databases, in *Proc. ACM SIGMOD* (1991), pp. 278–287
34. R.J. Lipton, J.F. Naughton, Query size estimation by adaptive sampling, in *Proc. Ninth PODS* (1990), pp. 40–46

35. R.J. Lipton, J.F. Naughton, D.A. Schneider, Practical selectivity estimation through adaptive sampling, in *Proc. ACM SIGMOD* (1990), pp. 1–11
36. R.J. Lipton, J.F. Naughton, D.A. Schneider, S. Seshadri, Efficient sampling strategies for relational database operations. *Theor. Comput. Sci.* **116**, 195–226 (1993)
37. K.D. Seppi, J.W. Barnes, C.N. Morris, A Bayesian approach to database query optimization. *ORSA J. Comput.* **5**, 410–419 (1993)
38. J.M. Hellerstein, P.J. Haas, H.J. Wang, Online aggregation, in *Proc. ACM SIGMOD* (1997), pp. 171–182
39. C. Jermaine, A. Dobra, S. Arumugam, S. Joshi, A. Pol, A disk-based join with probabilistic guarantees, in *Proc. ACM SIGMOD* (2005)
40. C.M. Jermaine, S. Arumugam, A. Pol, A. Dobra, Scalable approximate query processing with the DBO engine, in *Proc. ACM SIGMOD* (2007), pp. 725–736
41. C. Jermaine, A. Dobra, A. Pol, S. Joshi, Online estimation for subset-based SQL queries, in *Proc. 31st VLDB* (2005), pp. 745–756
42. G. Luo, C. Ellman, P.J. Haas, J.F. Naughton, A scalable hash ripple join algorithm, in *Proc. ACM SIGMOD* (2002), pp. 252–262
43. A. Pol, C. Jermaine, Relational confidence bounds are easy with the bootstrap, in *Proc. ACM SIGMOD* (2005)
44. P.G. Brown, P.J. Haas, BHUNT: automatic discovery of fuzzy algebraic constraints in relational data, in *Proc. 29th VLDB* (2003), pp. 668–679
45. I.F. Ilyas, V. Markl, P.J. Haas, P.G. Brown, A. Aboulmaga, CORDS: automatic discovery of correlations and soft functional dependencies, in *Proc. ACM SIGMOD* (2004), pp. 647–658
46. P. Brown, P. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, Y. Sismanis, Toward automated large-scale information integration and discovery, in *Data Management in a Connected World*, ed. by T. Härder, W. Lehner (Springer, New York, 2005)
47. M. Charikar, S. Chaudhuri, R. Motwani, V.R. Narasayya, Towards estimation error guarantees for distinct values, in *Proc. Nineteenth PODS* (2000), pp. 268–279
48. P.J. Haas, L. Stokes, Estimating the number of classes in a finite population. *J. Am. Stat. Assoc.* **93**, 1475–1487 (1998)
49. M. Wu, C. Jermaine, A Bayesian method for guessing the extreme values in a data set, in *Proc. 33rd VLDB* (2007), pp. 471–482
50. P. Billingsley, *Probability and Measure*, 2nd edn. (Wiley, New York, 1986)
51. A.M. Law, *Simulation Modeling and Analysis*, 4th edn. (McGraw-Hill, New York, 2007)
52. D.E. Knuth, *The Art of Computer Programming, vol. 2: Seminumerical Algorithms* (Addison-Wesley, Reading, 1969)
53. A.I. McLeod, D.R. Bellhouse, A convenient algorithm for drawing a simple random sample. *Appl. Stat.* **32**, 182–184 (1983)
54. J.S. Vitter, Random sampling with a reservoir. *ACM Trans. Math. Softw.* **11**, 37–57 (1985)
55. J.S. Vitter, Faster methods for random sampling. *Commun. ACM* **27**, 703–718 (1984)
56. M.T. Chao, A general purpose unequal probability sampling plan. *Biometrika* **69**, 653–656 (1982)
57. H. Brönnimann, B. Chen, M. Dash, P.J. Haas, Y. Qiao, P. Scheuermann, Efficient data reduction methods for on-line association rule discovery, in *Data Mining: Next Generation Challenges and Future Directions*, ed. by H. Kargupta, A. Joshi, K. Sivakumar, Y. Yesha (AAAI Press, Menlo Park, 2004)
58. B. Babcock, M. Datar, R. Motwani, Sampling from a moving window over streaming data, in *Proc. 13th SODA* (2002), pp. 633–634
59. T. Hagerup, C. Rub, A guided tour of Chernoff bounds. *Inf. Process. Lett.* **33**, 305–308 (1990)
60. W. Feller, *An Introduction to Probability Theory and Its Applications*, 3rd edn., vol. 1 (Wiley, New York, 1968)
61. R. Gemulla, W. Lehner, P.J. Haas, Maintaining bounded-size sample synopses of evolving datasets. *VLDB J.* **17**(2), 173–202 (2008)
62. R. Gemulla, W. Lehner, P.J. Haas, Maintaining Bernoulli samples over evolving multisets, in *Proc. Twenty Sixth PODS* (2007), pp. 93–102

63. G. Cormode, S. Muthukrishnan, I. Rozenbaum, Summarizing and mining inverse distributions on data streams via dynamic inverse sampling, in *Proc. 31st VLDB* (2005), pp. 25–36
64. G. Frahling, P. Indyk, C. Sohler, Sampling in dynamic data streams and applications, in *Proc. 21st ACM Symp. Comput. Geom.* (2005), pp. 142–149
65. P.B. Gibbons, Distinct sampling for highly-accurate answers to distinct values queries and event reports, in *Proc. 27th VLDB* (2001), pp. 541–550

Data Stream Management

Processing High-Speed Data Streams

Garofalakis, M.; Gehrke, J.; Rastogi, R. (Eds.)

2016, VII, 537 p. 103 illus., 16 illus. in color., Hardcover

ISBN: 978-3-540-28607-3