Information Systems 39 (2014) 199–216

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Adaptive stratified reservoir sampling over heterogeneous data streams

Mohammed Al-Kateb*, Byung Suk Lee

Department of Computer Science, The University of Vermont, Burlington, VT 05405, United States

A R T I C L E I N F O

Available online 4 April 2012

Keywords:
Reservoir sampling
Stratified sampling
Data streams

A B S T R A C T

Reservoir sampling is a known technique for maintaining a random sample of a fixed size over a *data stream* of an unknown size. While reservoir sampling is suitable for applications demanding a sample over the whole data stream, it is not designed for applications in which an input stream is composed of sub-streams with *heterogeneous* statistical properties. For this class of applications, the conventional reservoir sampling technique can lead to a potential damage in the statistical quality of the sample because it does not guarantee the inclusion of a statistically sufficient number of tuples in the sample from each sub-stream. In this paper, we address this heterogeneity problem by *stratifying* the reservoir sample among the underlying sub-streams. This stratification poses two challenges. First, a fixed-size reservoir should be allocated to individual sub-streams *optimally*, specifically to have the stratified reservoir sample used to generate estimates at the level of either the whole data stream or the individual sub-streams. Second, the allocation should be adjusted *adaptively* if and when new sub-streams appear in or existing sub-streams disappear from the input stream and as their statistical properties change. We propose a novel adaptive stratified reservoir sampling algorithm designed to meet these challenges. An extensive performance study shows the superiority of the achieved sample quality and demonstrates the adaptivity of the proposed sampling algorithm.

1. Introduction

Sampling is the process of selecting members of a population to derive some estimates representing the entire population [1]. *Random sampling* is the most basic sampling scheme under which all possible samples of the same size have an equal chance to be selected. Since random sampling usually generates consistent and unbiased estimates of the population, it has been used in a wide range of application domains such as computationals geometry [2,3], graph optimization [4], knowledge discovery [5,6], approximate query processing [7–12], and data stream processing [13–16].

When the data subject to sampling comes in the form of a *stream*, two fundamental challenges occur. First, the size of a data stream is typically unknown in advance. Hence, the sample fraction (or sampling rate) cannot be determined before the sampling begins. Second, in most cases the data arriving in a stream cannot be stored. Consequently, the data must be processed sequentially in a single pass. A common technique to overcome these challenges is the *reservoir sampling* [17,18], which selects a random sample of a fixed size without replacement from a stream of an unknown size. The utility and efficiency of the reservoir sampling brought about its use in a wide range of database applications including clustering [19–21], data warehousing [22], spatial data

* Corresponding author. Tel.: +1 424 206 9607; fax: +1 802 656 0696.
E-mail addresses: malkateb@cs.uvm.edu (M. Al-Kateb), bslee@cs.uvm.edu (B.S. Lee).

0306-4379/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2012.03.005

management [23], and approximate query processing [8,10–12,24–26].

One key assumption in the reservoir sampling algorithm is that the sample always represents the *whole stream*. This assumption, however, is not valid for many stream applications. In those applications, an input stream is composed of *sub-streams* which correspond to different groups with significantly different statistical properties (i.e., *mean* and *variance*). (We refer to this class of data streams as *heterogeneous* data streams.) Moreover, those applications use a sample to derive estimates at the level of either the entire data stream or the individual sub-streams. An example of such an application is discussed in Section 2.

Using the conventional reservoir sampling technique over a *heterogeneous* data stream does not guarantee that every sub-stream will have a statistically sufficient number of tuples in the reservoir. That is, the statistical quality of the sample is compromised for some sub-streams. Moreover, the technique maintains *one* random sample of a fixed size from *all* tuples seen so far in an input stream. Therefore, it is not suitable when multiple random sub-samples are needed to obtain the estimates of individual sub-streams.[1]

A concern for such heterogeneity naturally leads to the idea of *stratified sampling* [1]. Stratified sampling is designed to take a sample from a population made of heterogeneous groups. Specifically, the population is clustered into disjoint homogenous groups (or strata), and then a sample is taken randomly from each group (or stratum).[2] To the best of our knowledge, while stratified sampling has been used in the context of database systems (e.g., [27,28]), in no existing work have *data streams* been the target of a stratified sampling algorithm.

The streaming nature of the heterogeneous data introduces the following additional challenges. First, usually neither the number of sub-streams nor their statistical properties are known in advance. So, it is not possible to optimally allocate a stratified sample to sub-streams prior to sampling. Second, the membership of a data stream and the statistical properties of the member sub-streams may change over time. So, the allocation should have the ability to adapt to these changes.

This paper addresses the problem of maintaining a *stratified reservoir* sample over *heterogeneous* data streams for applications that demand reliable estimates at the level of either the *whole data stream* or *individual sub-streams*. There are two technical issues in this problem. The first issue is to allocate a given fixed-size reservoir optimally among sub-streams. To solve this issue, we adopt a statistical method known as the *power allocation* [29]. By controlling what is called the *power* parameter, this method allows for allocating a given sample size optimally [29] whether the estimates are required from the whole data stream or from the individual sub-streams. The second issue is to adjust the allocation as the data stream membership changes (i.e., new sub-streams appear or existing sub-streams disappear) or their statistical properties change over time. To solve this issue, the uniformity of the sample of each sub-stream should be maintained as the corresponding sample size changes over time. For this we use a simple variation of the *adaptive-size reservoir sampling* technique (from our prior work) [30] which maintains the uniformity of a reservoir sample with a required degree of confidence after the reservoir size is adjusted in the middle of sampling. We call the algorithm solving these two issues the *adaptive stratified reservoir sampling (ASRS)* algorithm.

We evaluate the proposed ASRS algorithm through two sets of experiments with respect to the achieved sample quality and the adaptivity of the algorithm. More specifically, in the first set of experiments, the sample quality (i.e., precision and accuracy[3]) is compared against the conventional reservoir sampling (RS) algorithm and a basic stratified reservoir sampling (BSRS) algorithm for different number of input sub-streams and for varying degree of heterogeneity among the sub-streams. The experiment results show that our proposed algorithm outperforms the RS algorithm by nearly an order of magnitude and achieves about twice higher sample quality than the BSRS algorithm. In the second set of experiments, we examine how the proposed algorithm adaptively adjusts the reservoir allocation. The results show that the algorithm adjusts the sub-sample sizes as the membership of a data stream and CVs of the member sub-streams change over time.

The contributions from this paper can be summarized as follows. First, it identifies stratified reservoir sampling over heterogeneous data streams as an important problem with real needs. Second, it proposes stratification as the key approach, and presents an algorithm for maintaining a stratified reservoir sample adaptively over a heterogeneous data stream while flexibly assuming the sample can be used for one estimate from the whole stream or multiple estimates from individual sub-streams. Third, through experiments it demonstrates the superiority of the achieved sample quality and the adaptivity of the proposed algorithm.

The rest of the paper is organized as follows. Section 2 discusses a motivating example application. Section 3 provides an overview of the reservoir sampling and stratified sampling techniques. Section 4 formulates the research problem and presents the proposed adaptive stratified reservoir sampling algorithm. Section 5 presents performance study and discusses experiment results. Section 6 reviews related work. Section 7 concludes this paper and suggests future work.

2. Motivating example

The Federal Communications Commission (FCC) runs an auction system [31] through which auctions for

[1] Terminology: we say that a *sub-sample* is produced from a *sub-stream* and is stored in a *sub-reservoir*.

[2] The statistical properties (i.e., mean and variance) of each stratum is often replaced by the *coefficient of variation* (*CV*), defined as the ratio of the standard deviation to the mean.

[3] Sample accuracy is the degree of closeness of the estimate to its true value, whereas sample precision is the degree to which the estimates from different samples taken from the same data set vary from one another (see Fig. 5 in Section 5 for an illustration).

obtaining the licenses of electromagnetic spectrum are conducted electronically over the Internet. Any prospective bidder can submit an application and, once approved as a qualified bidder, can place bids any time via a web browser and follow the progress and outcome of each auction.

In this auction system, bids placed on multiple concurrent auctions form a stream. This stream is composed of *multiple sub-streams* each of which represents the bidding records of a particular auction. The auction data stream is *heterogeneous*, as the mean and the variance of bidding amounts vary significantly from one auction to another depending on the type of an auction item. Table 1 and Fig. 1 illustrate this heterogeneity with respect to two selected FCC auction datasets (i.e., sub-streams). The Guard Band is a lower price item with no bidding amount higher than \$100K and with the mean and standard deviation of all bidding amounts around \$74K and \$24.6K, respectively. In contrast, the Broadband Personal Communication Service (PCS) is a higher price item with no bidding amount lower than \$100K and with the mean and standard deviation of all bidding amounts around \$5.7M and \$9M, respectively. (Interestingly, the distributions of the bidding amounts are opposite – they are skewed to the higher end of the range for the Guard Band and to the lower end of the range for the Broadband PCS.)

From a practical viewpoint, the scope of sampling is twofold in this auction system. On one hand, for an aggregated analysis of all auction items, a sample is needed from all bidding amounts in the *entire* auction data stream – for example, to estimate the median of bidding amounts across all auction items. On the other hand, for an itemized analysis of auction items, a sample is needed from the bidding amounts in *individual* auction data streams to generate the estimate for each auction. This dual sampling-scope is a feature appreciated in many other applications as well.

Table 1
Mean and variance of bidding amounts (US dollars) in two FCC auctions [31].

Auction Item	Mean	Standard deviation	Variance
FCC 700 MHz Guard Band	73,964	24,591	6.0×10^8
FCC Broadband PCS	5,737,987	9,001,320	8.1×10^{13}

3. Background

This section provides the necessary backgrounds on reservoir sampling and stratified sampling.

3.1. Reservoir sampling

Algorithm 1. Conventional reservoir sampling (RS).

```
Require |r| // the size of a reservoir r
1:  $k=0$;
2:  for each tuple $s_k$ arriving from the input stream do
3:    $k=k+1$;
4:    if $k \le |r|$ then
5:      $r[k-1]=s_k$; // add the tuple to the reservoir
6:    else
7:      $n_1$=rand(0,1); // generate a random number between 0 and 1
8:      if $n_1 \le \frac{|r|}{k}$ then
9:        $n_2$=rand(0, $|r|-1$); // generate a random number between 0 and $|r|-1$
10:       $r[n_2]=s_k$; // replace a randomly selected tuple in reservoir with the accepted tuple
11:     end if
12:   end if
13: end for
```

Reservoir sampling [17,18] is a technique for selecting a uniform random sample of a fixed size without replacement from an input stream of an unknown size. Let r denote a reservoir of size $|r|$ tuples and let k denote the number of tuples seen so far from the input stream. Initially, the conventional reservoir sampling algorithm (see Algorithm 1) places all tuples in the fixed-size reservoir r until the reservoir becomes full (Lines 4 and 5). After that, each kth tuple is accepted for inclusion in the reservoir with the probability of $|r|/k$ and an accepted tuple replaces a randomly selected tuple in the reservoir (Lines 6–12).

Fig. 2 gives an illustration of the conventional reservoir sampling algorithm. Fig. 2(a) shows the reservoir of three tuples after being filled with the first three tuples – s_1, s_2, and s_3 – from the input stream S. At this point, all three tuples have the same probability of 1 to be in the reservoir. Fig. 2(b) shows that the fourth tuple, s_4,

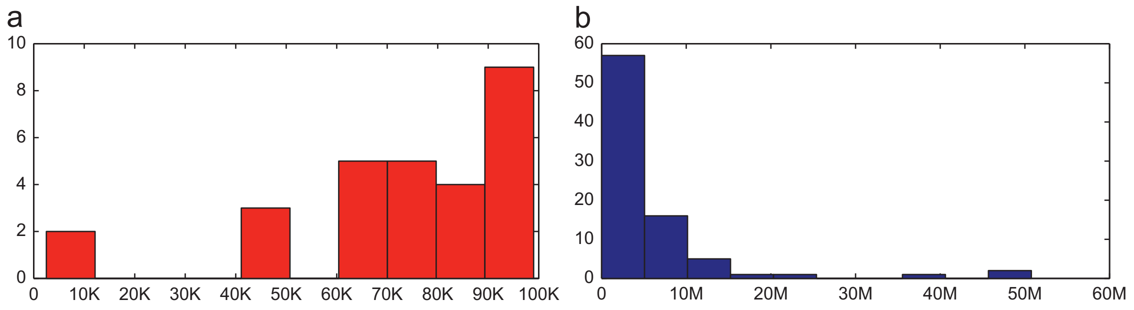

Fig. 1. Histograms of the bidding amounts of two FCC auctions. (a) FCC 700 MHz Guard Band auction and (b) FCC Broadband PCS auction.

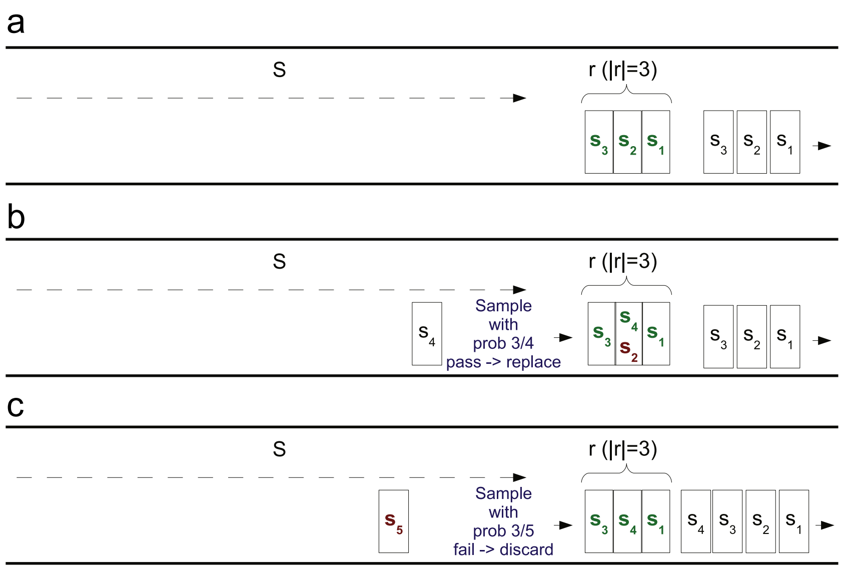

Fig. 2. An illustration of the conventional reservoir sampling. (a) Upon the arrival of tuple s_3. (b) Upon the arrival of tuple s_4 and (c) Upon the arrival of tuple s_5.

is sampled with the probability 3/4 and accepted to be included in the reservoir, randomly replacing one of the three tuples already in the reservoir (in this example it is the tuple s_2). Fig. 2(c) shows that the fifth tuple, s_5, is sampled with the probability 3/5, but it is not accepted to be included in the reservoir.

This reservoir sampling algorithm guarantees that the reservoir always holds a *uniform* random sample of the k tuples seen so far [17]. After the kth tuple arrives, each of the k tuples has the equal probability $|r|/k$ to be included in the reservoir. We can use induction to prove the uniformity of conventional reservoir sampling as shown below.

Proof (by induction). Base case ($k \le |r|$): This case is trivial. Since all tuples are accepted until the reservoir is full, all of the first k ($\le |r|$) tuples have the same probability 1 to be in the reservoir.

Induction case ($k > |r|$): Assume the induction hypothesis that, upon the arrival of the kth tuple, each of the k tuples has the same probability $|r|/k$ to be in the reservoir. Now, it needs to be shown that, upon the arrival of the next (i.e., $(k+1)$th) tuple, each of the $k+1$ tuples has the same probability $|r|/(k+1)$ to be in the reservoir. This can be shown as follows.

The probability of the $(k+1)$th tuple to be in the reservoir is $|r|/(k+1)$ by the design of the algorithm. The probability of any other arbitrary tuple s_i ($i=1,2,\ldots,k$) to be in the reservoir is $|r|/k$ (from the induction hypothesis) multiplied by the sum of the probabilities of the following two cases.

Case 1: The $(k+1)$th tuple is not accepted for inclusion in the reservoir. The probability of this case is $1-(|r|/(k+1))$.

Case 2: The $(k+1)$th tuple is accepted for inclusion in the reservoir but some other tuple than s_i is selected to be replaced. The probability of this case is $(|r|/(k+1))(|r-1|/|r|)$.

Adding these two probabilities gives the following probability:

$$\frac{|r|}{k}\left(\left(1-\frac{|r|}{k+1}\right)+\left(\frac{|r|}{k+1}\frac{|r-1|}{|r|}\right)\right)=\frac{|r|}{k+1} \qquad \square$$

3.2. Stratified sampling

Stratified sampling [1] is a sampling scheme in which a heterogeneous population R is initially clustered into n disjoint homogeneous strata, $R_1, R_2, \ldots, R_i, \ldots, R_n$, and then a sample r_i is taken randomly from each stratum R_i. Every member of R should belong to one and only one stratum (i.e., $R_i \cap R_j = \phi (i \ne j)$ and $R_1 \cup R_2 \cup \cdots \cup R_i \cup \cdots \cup R_n = R$). A stratified sample of a given size is expected to have higher statistical precision (i.e., lower sampling error) than a random sample of the same size taken from the same population when the statistical properties (i.e., mean and variance) of strata vary considerably.

Allocating a given sample size $|r|$ to different strata is a fundamental issue in stratified sampling. Obviously, the allocation is subject to the maximum $|r|$ constraint on the sum of sub-sample sizes assigned to individual strata, $|r_1|, |r_2|, \ldots, |r_n|$:

$$\sum_{i=1}^{n} |r_i| \le |r| \qquad (1)$$

There are two allocation methods commonly used for a stratified sample, the *proportional* allocation [1] and the *Neyman* allocation [32]. Under the *proportional* allocation, the sample size of each stratum is determined in proportion to the size of the stratum:

$$|r_i| = |r| \times \frac{|R_i|}{|R|} \tag{2}$$

where R denotes the whole population, R_i denotes a stratum, and $|R|$ and $|R_i|$ denote the sizes of R and R_i, respectively. Under the *Neyman* allocation, the sample size of each stratum is determined in proportion to the standard deviation as well as the size of the stratum:

$$|r_i| = |r| \times \frac{\sigma_i |R_i|}{\sum_{j=1}^{n} \sigma_j |R_j|} \tag{3}$$

where σ_i denotes the standard deviation of R_i.

4. Adaptive stratified reservoir sampling

The proposed adaptive stratified reservoir sampling algorithm is described in this section. As mentioned in Section 1, there are two technical issues handled by the proposed algorithm: (1) determining the optimal sizes of sub-samples for each sub-stream and (2) maintaining the uniformity of each sub-sample as its size changes. In this section, we first formulate the problem formally in Section 4.1 and discuss our approaches to the two technical issues in Sections 4.2 and 4.3, respectively, and then summarize them into one algorithm in Section 4.4.

4.1. Problem formulation

The problem of allocating a fixed-size reservoir to sub-streams is an adaptive optimization problem formulated as follows. An input data stream S consists of n sub-streams $S_1, S_2, \ldots, S_n$. Each sub-stream $S_i (i = 1, 2, \ldots, n)$ is a sequence of tuples $s_{i_1}, s_{i_2}, \ldots$ such that $S_i \cap S_j = \phi \ (i \neq j)$ and $\bigcup_{S_i} = S$. Given a total available size of $|r|$ tuples in a reservoir r, the objective is to allocate $|r|$ *optimally* among the n sub-streams subject to the following constraint at any point in time t:

$$\sum_{i=1}^{n} |r_i(t)| \leq |r| \tag{4}$$

where $r_i(t)$ denotes the sample allocated for S_i at time point t and $|r_i(t)|$ denotes its size. The optimality criterion is the sample quality, and there is some minor difference in the specific criteria depending on which purpose (i.e., one whole sample or individual sub-samples) the sample is used for (details in Section 4.2).

Fig. 3 illustrates the processing of the adaptive stratified reservoir sampling. It shows that at time t_1, the reservoir r is divided into r_1 of size 3 for tuples that belong to sub-stream S_1, r_2 of size 2 for tuples that belong to sub-stream S_2, and r_3 of size 3 for tuples that belong to sub-stream S_3. At time t_2, the sizes of r_1, r_2, and r_3 have decreased, increased, and decreased, respectively, while the total size of the reservoir r remains the same.

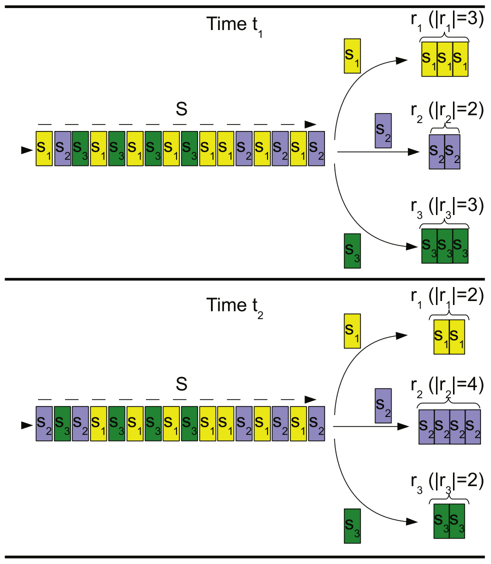
Time t1
S
S1 S2 S3 S1 S3 S1 S3 S1 S3 S1 S1 S2 S1 S2 S1 S2
S1
S2
S3
r1 (|r1|=3)
r2 (|r2|=2)
r3 (|r3|=3)
Time t2
S
S2 S3 S2 S1 S3 S1 S3 S1 S3 S1 S1 S2 S1 S2 S1 S2
r1 (|r1|=2)
r2 (|r2|=4)
r3 (|r3|=2)

Fig. 3. An illustration of adaptive stratified reservoir sampling.

4.2. Optimal allocation of the stratified reservoir to sub-samples

For the flexible aim of generating estimates from the whole sample or from separate sub-samples, the commonly used *Neyman allocation* is not adequate enough since it is geared for the former case only. To overcome this limit, we adopt another statistical method known as *power allocation* [29]. The power allocation method provides a way to allocate the sample to different strata whether the sample is used to generate a single estimate for the underlying population as a whole or multiple estimates separately for each of the underlying strata. This flexibility is enabled by a control parameter called the *power* of allocation.

Formally, the size of a sample, $|r_i|$, assigned to a stratum R_i is computed as

$$|r_i| = |r| \times \frac{\sigma_i \left(\sum_{j=1}^{|R_i|} y_{ij}\right)^q \Big/ \frac{\sum_{j=1}^{|R_i|} y_{ij}}{|R_i|}}{\sum_{k=1}^{n} \sigma_k \left(\sum_{j=1}^{|R_k|} y_{kj}\right)^q \Big/ \frac{\sum_{j=1}^{|R_k|} y_{kj}}{|R_k|}} \tag{5}$$

where y_{ij} denotes the sampling attribute value of the jth member in R_i, σ_i denotes the standard deviation of the sampling attribute values in R_i, and q denotes the power of allocation.

When the stratified sample is used of the *entire population*, power allocation aims to minimize the sampling variance of the estimator of the whole stratified sample, where the sampling variance is formulated as

$$\sum_{i=1}^{n} \sigma_i \frac{|R_i|(|R_i| - |r_i|)}{|r_i|} \tag{6}$$

In this case, it achieves an optimal allocation by setting the power value to 1. Note that this results in the exact

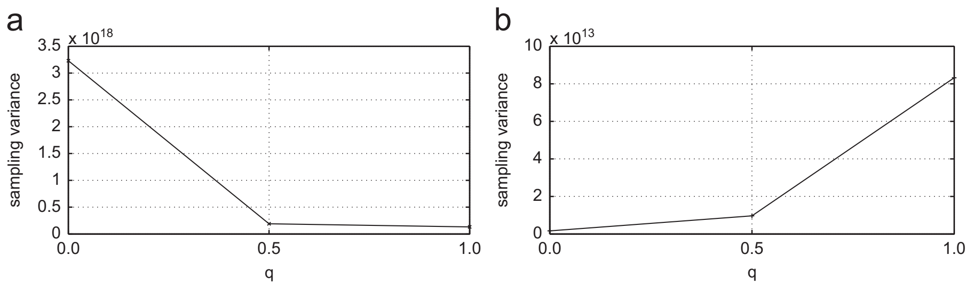

Fig. 4. The effect of the value of the power q on sampling variance. (a) The entire stream of all FCC auctions and (b) A single sub-stream (for FCC Broadband PCS auction).

Neyman allocation (see Eq. (3)), that is

$$|r_i| = |r| \times \frac{\sigma_i(\sum_{j=1}^{|R_i|} y_{ij})^1 \Big/ \frac{\sum_{j=1}^{|R_i|} y_{ij}}{|R_i|}}{\sum_{k=1}^{n} \sigma_k(\sum_{j=1}^{|R_k|} y_{kj})^1 \Big/ \frac{\sum_{j=1}^{|R_k|} y_{kj}}{|R_k|}} = |r| \times \frac{\sigma_i |R_i|}{\sum_{k=1}^{n} \sigma_k |R_k|} \quad (7)$$

When the stratified sample is used at the level of *individual strata*, Neyman allocation may cause the sampling variances of some strata to be larger than those achievable by considering strata individually. For an illustration, let us examine the sampling variance achieved using the Neyman allocation method when the sample is used at the level of the entire stream and at the level of the individual strata. For this purpose, we use the FCC auction example (introduced in Section 1). Fig. 4 shows the sampling variance of a sample taken from the auction bid streams when the power allocation method is used with three different power values, $q = 0.0$, 0.5, and 1.0. Note that the Neyman allocation is identical to the power allocation for $q = 1$ (see Eq. (7)). Fig. 4(a) shows that the Neyman allocation indeed produces the smallest sampling variance among the three power-value cases when the entire stream is considered. In contrast, Fig. 4(b) shows that the Neyman allocation does cause the sampling variance of some sub-streams (the FCC Broadband PCS auction sub-stream in this case) to be the largest among the three power-value cases – larger by an order of magnitude.

Power allocation's remedy for this deficiency of the Neyman allocation is to allocate sub-sample sizes in proportion to CV (i.e., the ratio of the standard deviation to the mean) of each stratum, which is achieved by setting the power to 0. In this case,

$$|r_i| = |r| \times \frac{\sigma_i\left(\sum_{j=1}^{|R_i|} y_{ij}\right)^0 \Big/ \frac{\sum_{j=1}^{|R_i|} y_{ij}}{|R_i|}}{\sum_{k=1}^{n} \sigma_k\left(\sum_{j=1}^{|R_k|} y_{kj}\right)^0 \Big/ \frac{\sum_{j=1}^{|R_k|} y_{kj}}{|R_k|}}$$

$$= |r| \times \frac{\sigma_i \Big/ \frac{\sum_{j=1}^{|R_i|} y_{ij}}{|R_i|}}{\sum_{k=1}^{n} \sigma_k \Big/ \frac{\sum_{j=1}^{|R_k|} y_{kj}}{|R_k|}} \quad (8)$$

Applying this power allocation to data stream gives the following formula for determining sub-sample sizes at any point in time t:

$$|r_i(t)| = |r| \times \frac{\sigma_i(t)\left(\sum_{j=1}^{|S_i(t)|} y_{ij}\right)^q \Big/ \frac{\sum_{j=1}^{|S_i(t)|} y_{ij}}{|S_i(t)|}}{\sum_{k=1}^{n} \sigma_k(t)\left(\sum_{j=1}^{|S_k(t)|} y_{kj}\right)^q \Big/ \frac{\sum_{j=1}^{|S_k(t)|} y_{kj}}{|S_k(t)|}} \quad (9)$$

where $|r_i(t)|$ denotes the size of a sub-sample allocated for a sub-stream S_i at time point t, $\sigma_i(t)$ denotes the running standard deviation of the sampling attribute values in S_i up to t, and $|S_i(t)|$ denotes the number of tuples processed from S_i up to t. Note that a *running* standard deviation is required for the calculations over a data stream. For this, we use an efficient recurrence relation [33], known as the *updating* method which is capable of calculating the standard deviation in a single scan of the data and providing precise calculation even when the data values are large. Details on this method can be found in Appendix A.

4.3. *Maintaining the uniformity of each sub-sample*

Since the size of a sub-sample may change as a result of the optimal allocation, the uniformity of each sub-sample must be maintained across the change of size. To this end, we use the notion of uniformity confidence (*UC*), introduced in our prior study [30], and use it as the basis for designing a reservoir sampling algorithm which can adapt to the change of a sub-sample size.

4.3.1. *Uniformity confidence*

Uniformity confidence refers to the probability that a sampling algorithm generates a uniform random sample after the sample size changes in the middle of sampling. It is based on the definition that a sample is said to be a uniform random sample if produced by a sampling algorithm in which all statistically possible samples of the same size are equally likely to be selected; in this case, we say the uniformity confidence in the sampling algorithm equals 100%. In contrast, if some of the statistically possible samples cannot be selected using a sampling algorithm, then we say the uniformity confidence in the sampling algorithm is less than 100%. Thus, the uniformity confidence is defined as follows:

$$\frac{\text{The number of different samples of the same size possible with the algorithm}}{\text{The number of different samples of the same size possible statistically}} \times 100 \tag{10}$$

Specifically for reservoir sampling, the uniformity confidence is defined as the probability that the produced sample is a uniform random sample of *all the tuples seen so far*. That is, if k tuples have been seen so far, then the uniformity confidence is 100% if and only if all statistically possible reservoir samples of the same size have an equal probability to be selected from the k tuples.

4.3.2. *Adaptive reservoir sampling algorithm*

The theoretical study in [30] concludes that if the reservoir size decreases then the sample uniformity can be maintained in the *reduced* reservoir with 100% confidence by randomly evicting tuples from the original reservoir. It also concludes that if the reservoir size increases then it is not possible to attain 100% confidence in the *enlarged* reservoir but it is possible to ensure the uniformity confidence above a given threshold by sampling from upcoming tuples in the input stream. Appendix B provides additional details with theorems.

For maintaining the uniformity confidence for an enlarged reservoir, one seemingly intuitive approach would be to just fill up the added room in the reservoir from the incoming tuples, with all the existing tuples retained. The resulting uniformity confidence can be expressed as follows:

$$UC(k,|r|,\delta,m) = \frac{\binom{k}{|r|}\binom{m}{\delta}}{\binom{k+m}{|r|+\delta}} \times 100 \tag{11}$$

where δ is the amount of increased reservoir size and m $(\geq \delta)$ is the number of upcoming tuples from which the increased reservoir is filled. Unfortunately, however, this approach has the side effect of producing a *stratified* sample (of size $|r|+\delta$) consisting of two strata: one stratum (of size $|r|$) obtained by sampling k tuples randomly with the sampling probability $|r|/k$ and another stratum (of size δ) obtained by sampling m tuples randomly with the sampling probability δ/m. Undoubtedly, stratifying a single sub-sample is statistically inadequate.

Algorithm 2. Adaptive-size reservoir sampling (ARS).

```
Require: $|r|$ // original size of a reservoir r
         $k$ // number of tuples seen so far
         $\zeta$ // uniformity confidence threshold
1:  while true do
2:    calculate $|r_{new}|$ // new size of a reservoir r
3:    while $|r| == |r_{new}|$ do
4:      continue sampling using RS (i.e., Algorithm 1);
5:    end while
6:    if $|r_{new}| < |r|$ then
7:      // i.e., reservoir size is decreased by $\delta$
8:      $\delta = |r| - |r_{new}|$
9:      randomly evicts $\delta$ tuples from the reservoir;
10:   else
11:     // i.e., reservoir size is increased by $\delta$
12:     $\delta = |r_{new}| - |r|$
13:     find the minimum value of $m$ (Eq. (12)) such that $UC \geq \zeta$;
14: flip a biased coin to decide on $x$, the number of tuples to retain
    in the reservoir (Eq. (13));
15:     randomly evict $|r| - x$ tuples from the reservoir;
16:     select $\delta + |r| - x$ tuples from the incoming $m$ tuples using RS;
17:   end if
18: end while
```

Algorithm 2, also introduced in our prior work [30], has no such a drawback and achieves better uniformity. The steps of the algorithm are as follows. First, it finds the minimum number (m) of incoming tuples that should be considered to refill the enlarged reservoir such that the resulting uniformity confidence (calculated using Eq. (12)) exceeds the given threshold (Line 13). Then, it decides probabilistically (using Eq. (13)) the number (x) of tuples to retain in the enlarged reservoir (Line 14) and randomly evicts the remaining number ($|r|-x$) of tuples (Line 15). Then, it fills the room available in the enlarged reservoir from the incoming tuples (Line 16):

$$UC(k,|r|,\delta,m) = \frac{\sum_{x=\max\{0,(|r|+\delta)-m\}}^{|r|} \binom{k}{x}\binom{m}{|r|+\delta-x}}{\binom{k+m}{|r|+\delta}} 100 \tag{12}$$

$$p(x) = \frac{\binom{k}{x}\binom{m}{|r|+\delta-x}}{\binom{k+m}{|r|+\delta}} \tag{13}$$

In this paper, we use a simple variation of Algorithm 2 (ARS) in which the number of incoming tuples required to refill an enlarged reservoir is computed as follows:

$$m = \frac{\delta \times k}{|r|} \tag{14}$$

The rationale for computing the value of m this way is a simple heuristic that, since r has been filled from k tuples so far, the room for additional δ tuples should be filled in proportion to $k/|r|$, that is, $\delta k/|r|$ tuples. This heuristic facilitates the use of the algorithm by eliminating the need to conduct an expensive search to find the optimum value of m using Eq. (12), which is an inverse-mapping problem.

4.4. *Adaptive stratified reservoir sampling algorithm*

Based on the discussions above, our adaptive stratified reservoir sampling algorithm works as shown in Algorithm 3. In this algorithm, the input stream S is treated as a *set* of sub-streams S_1, S_2, etc. and ALG_i refers to the sampling algorithm – either Algorithm 1 (RS) or Algorithm 2 (ARS) – currently in use for the sub-stream S_i.

Algorithm 3. Adaptive stratified reservoir sampling (ASRS).

```
Require: $|r|$ // original size of a reservoir r
         $q$ // power of allocation
         $\Delta$ // sample reallocation time interval
    // **************INITIALIZATION PHASE**************
1:  for each new tuple $s$ arriving from a sub-stream $S_i$ do
2:    if reservoir $r$ is not full then
3:      add $s$ to $r$;
4:      update the running statistics of $S_i$;
```

```
5:        if $S_i \notin S$ // i.e., $S_i$ is a new sub-stream then
6:          $S = S \cup \{S_i\}$;
7:          $ALG_i$=RS; // start sampling using CRS
8:        end if
9:      else
10:       break; // go to line 13
11:     end if
12:   end for
13:   for each $S_i \in S$ do
14:     $|r_i| = \text{size}(S_i)$; // initialize sub-reservoir sizes
15:   end for
      // *****************SAMPLING PHASE*****************
16:   while true do
17:     for each new tuple $s$ arriving from a sub-stream $S_i$ do
18:       if $S_i \notin S$ // i.e., $S_i$ is a new sub-stream then
19:         $S = S \cup \{S_i\}$;
20:         $ALG_i$=RS; // start sampling using Algorithm 1
21:       end if
22:       sample $s$ into the sub-reservoir $r_i$ using $ALG_i$; // either RS or ARS
23:       update the running statistics of $S_i$;
24:       if the time interval $\Delta$ has passed then
25:         break; // go to line 28 to calculate sub-reservoir sizes
26:       end if
27:     end for
28:     for each sample $r_i$ allocated to sub-stream $S_i$ do
29:       if $S_i$ expires from $S$ // e.g., due to a punctuation then
30:         $S = S - \{S_i\}$;
31:         $|r_i(t)| = 0$;
32:       else
33:         calculate $|r_i(t)|$ for $S_i$ using Eq. (9) with the given value of $q$;
34:         if $|r_i(t)|$ has changed as a result then
35:           $ALG_i$=ARS; // Algorithm 2
36:         else
37:           $ALG_i$=RS; // Algorithm 1
38:         end if
39:       end if
40:     end for
41:   end while
```

In the initialization phase of the algorithm (Lines 1–15), the first $|r|$ tuples in a data stream S are added to the reservoir while the running statistics of sub-streams are being updated (Lines 3 and 4). The sampling starts using RS for all new sub-streams (Lines 5–8) and, once the reservoir becomes full, the size of a sub-reservoir is initialized in proportion to the number of tuple seen so far from the corresponding sub-stream (Lines 13–15).

In the sampling phase (Lines 16–41), each time a new tuple s arrives from a sub-stream S_i, the algorithm decides to sample s using RS if S_i is a new sub-stream (Lines 18–21). Then, the algorithm samples s into r_i using the corresponding sampling algorithm (i.e., either RS or ARS) while updating its running statistics (Lines 22 and 23 and 28–40). Periodically, the algorithm reallocates the reservoir size optimally among sub-streams (Lines 24–26). Specifically, if a sub-stream has expired from the input stream (e.g., due to the presence of a punctuation), the memory of the sub-reservoir of that sub-stream is released (Lines 29–31). Otherwise, the algorithm calculates the optimal sample size for the sub-stream (Line 33). If the size of r_i changes as a result, then the algorithm switches over to ARS to continue sampling the incoming S_i tuples (Lines 34 and 35). Note that ARS quickly resumes RS once the size adjustment is handled. If the size of r_i does not change, then the algorithm samples the incoming S_i tuples using RS (Line 37).

5. Performance evaluation

We conduct two sets of experiments. The first set of experiment evaluates the performance of the *adaptive stratified reservoir sampling* (*ASRS*) algorithm against the *conventional reservoir sampling* (*RS*) algorithm and a *basic stratified reservoir sampling* (*BSRS*) algorithm which uses the proportional allocation method [1], with respect to the sample quality. The second set of experiments demonstrates the adaptivity of the ASRS to the changes of data stream membership and the statistical characteristics of member sub-streams. In this section, the design and setup of the experiments are described in Section 5.1 and the results of the experiments are presented in Section 5.2.

5.1. Experiment design and setup

Intuitively, two factors affect the performances of algorithms over a data stream consisting of multiple heterogeneous sub-streams: the number of sub-streams and the degree of heterogeneity among the sub-streams. These two parameters are thus used in the comparisons between ASRS, RS, and BSRS.

5.1.1. Performance metrics

The two kinds of sample quality mentioned in Section 1 are used to compare the performances of ASRS, RS, and BSRS: *accuracy* and *precision* (see Fig. 5 for an illustration). Sample accuracy is a measure of how close the estimated value is to its true value. Sample precision is a measure of how close the estimates measured from different samples taken from the same set of data are to one another. Specifically, we use the *error in estimated mean* (*EEM*), the difference between the mean value estimated from the sample and the actual mean value, as the metric of sample accuracy. The estimated mean for a random sample is

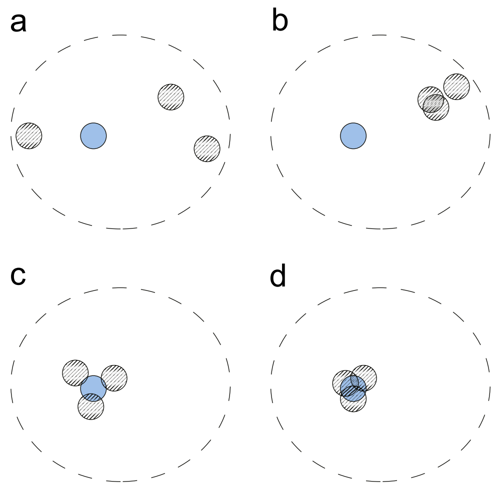

Fig. 5. An illustration of the performance metrics: accuracy vs precision. (a) Low accuracy, low precision. (b) Low accuracy, high precision. (c) High accuracy, low precision and (d) High accuracy, high precision. The solid circle indicates the actual value and the shaded circles indicate estimated values.

calculated as

$$\frac{1}{|r|}\sum_{i=1}^{|r|} y_i \tag{15}$$

where y_i denotes the value of the sampling attribute of the ith tuple in a sample r [1]. Extended from it, the estimated mean for a stratified sample is calculated as

$$\sum_{i=1}^{n}\frac{|S_i|}{|S|}\left(\frac{1}{|r_i|}\sum_{j=1}^{|r_i|} y_{ij}\right) \tag{16}$$

where y_{ij} denotes the value of sampling attribute of the jth tuple in a sub-sample r_i [1].

On the other hand, we use the *standard error* (*SE*), a common statistical quantification of the sample precision, as the metric of sample precision. The SE is a measure of how precise the sample is; the larger the SE, the lower the statistical precision of the sample is, and vice versa. The SE for a random sample is computed as

$$\sqrt{\left(1-\frac{|r|}{|S|}\right)\frac{\sigma^2}{|S|}} \tag{17}$$

where σ^2 denotes the variance of the entire sample [1]. Extended from it, the SE for a stratified sample is computed as

$$\frac{1}{|S|}\sqrt{\sum_{i=1}^{n}|S_i|^2\left(1-\frac{|r_i|}{|S_i|}\right)\frac{\sigma_i^2}{|r_i|}} \tag{18}$$

where σ_i^2 denotes the variance of the ith sub-sample [1].

5.1.2. Datasets

Experiments are conducted using both synthetic and real datasets. Synthetic datasets are used to examine the effect of the statistical characteristics of an input data stream on the quality of the sample. Now, we describe the process of synthetic dataset generation and outline the profile of the real datasets.

Synthetic datasets

A synthetic data stream is generated bottom up, that is, by first generating sub-streams and then combining them to form one stream. The sampling attribute value in each sub-stream S_i has the *doubly-truncated normal distribution* [34], i.e., the normal distribution with bounded lower and upper ends. Formally, if a random variable $X \backsim N(\mu,\sigma)$ has the normal distribution such that $-\infty \le l \le X \le u \le \infty$, then X is considered to have a doubly-truncated normal distribution with the probability density function:

$$pdf(x;\mu,\sigma,l,u) = \frac{\frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right)}{\Phi\left(\frac{u-\mu}{\sigma}\right)-\Phi\left(\frac{l-\mu}{\sigma}\right)} \tag{19}$$

where $\phi(x)$ is the probability density function of the standard normal distribution (i.e., $\phi(\mathbf{x}) = (\mathbf{1}/\sigma\sqrt{\mathbf{2}\pi})\,\mathbf{e}^{-(x-\mu)^2/2\sigma^2}$) and $\Phi(x)$ is its cumulative distribution function (i.e., $\Phi(\mathbf{x}) = \int_l^u \phi(\mathbf{x})\,\mathbf{dx}$) [34]. This distribution is used in many applications like inventory management and financial applications, in which the values are naturally constrained within a certain bound [35].

The datasets are synthesized from a different number of sub-streams (n) and with a varying degree of heterogeneity among the sub-streams (*DH*). *DH* is defined as the ratio of the *inter-sub-stream* variability to the *intra-sub-stream* variability. With the variability expressed in terms of *CV* [29], we define *DH* as the ratio of the *standard deviation* among the CVs of sub-streams ($\sigma_{[CV]}$) to the *average* of the CVs of sub-streams ($\mu_{[CV]}$). With the doubly truncated normal distribution in place, we know that the standard deviation of the sampling attribute values of a sub-stream S_i is bounded by half the range of these values. This means that each CV_i is bounded within the range of 0–1. Consequently, *DH* is also bounded within the range of 0–1.

Given the values of n and *DH*, the synthetic dataset generator works as follows. First, it sets the value of $\mu_{[CV]}$ to 0.5 (note $0<\mu_{[CV]}\le 1$) and calculates the value of $\sigma_{[CV]}$ accordingly ($\sigma_{[CV]} = DH \times \mu_{[CV]}$). Second, it generates n random numbers from a doubly truncated normal distribution with $\mu_{[CV]}$, $\sigma_{[CV]}$, $l_{[CV]} = \mu_{[CV]}-\sigma_{[CV]}$, and $u_{[CV]} = \mu_{[CV]}+\sigma_{[CV]}$. The n random numbers generated correspond to the *CV*s of the n sub-streams. Third, for S_i, the synthetic dataset generator uses the value of CV_i to assign the values of $\mu_{[S_i]}$ and $\sigma_{[S_i]}$ randomly such that $\sigma[S_i]/\mu_{[S_i]} = CV_i$. Finally, the generator produces the values of S_i from a doubly truncated normal distribution with $\mu_{[S_i]}$, $\sigma_{[S_i]}$, $l_i = \mu_{[S_i]}-\sigma_{[S_i]}$, and $u_i = \mu_{[S_i]}+\sigma_{[S_i]}$.

Fig. 6 shows an example of different datasets with varying degree of heterogeneity. In this example, the number of sub-streams is 10 and the values of *DH* are set to 30%, 50%, and 70%. When *DH* is relatively low (e.g., 30% in Fig. 6(a)), we see that most of the sub-streams have *wide* and *similar* spreads of sampling attribute values. The wide spread of each sub-stream indicates that the variability within each sub-stream is high, and the similar spreads among sub-streams indicates that the variability across sub-streams is low. These two combined indicate a low degree of heterogeneity in the entire stream. In contrast, when the *DH* is relatively high (e.g., 70% in Fig. 6(c)), we see that most sub-streams have narrow and dissimilar spreads of the sampling attribute values. This is the converse of Fig. 6(a) case above, and thus indicates a high degree of heterogeneity in the entire stream.

Real datasets

Two kinds of real datasets are used, one (SENS) in the wireless sensor networks application and one (AUCT) in the auction application.

- The sens real dataset is weather measurements from sensors deployed through the Intel Berkeley Research lab to gather time-stamped topology information, along with humidity, temperature, light and voltage values [36]. SENS is a projection of this data on two attributes, *sensor mote id* and *temperature measurement* acquired from 55 motes. (Data from three motes have incomplete readings and thus have been discarded.) SENS is characterized with a *low* degree of heterogeneity. The low degree of heterogeneity among the temperature

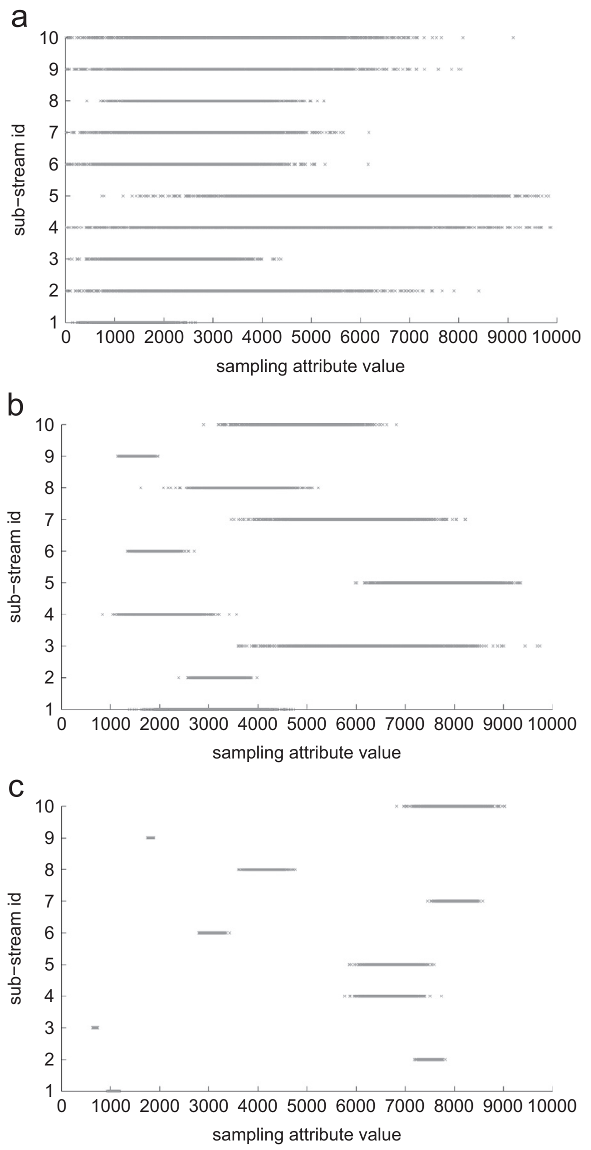

Fig. 6. Scatter plots of synthetic datasets with different degrees of heterogeneity. (a) Degree of heterogeneity = 30%. (b) Degree of heterogeneity = 50% and (c) Degree of heterogeneity = 70%.

readings of different motes is due to the fact that temperatures of nearby regions are expected to be close to each other.

- The AUCT real dataset is for auctions conducted over the Internet through the Federal Communications Commission (FCC) [31]. The entire dataset consists of 55 auction sub-datasets. Each sub-dataset contains bidding information of one auction. We have merged the 55 auction sub-datasets (from the bidding results of round 1) into one single dataset. The order of tuples in the resulting dataset is shuffled and the resulting tuples are projected on two attributes, *auction ID* and *bidding amount*. AUCT is characterized with a *high* degree of heterogeneity. The high degree of heterogeneity of the bidding amounts is intuitive since the bidding amounts can vary to a large extent depending on the auction item.

5.2. Experiment results

5.2.1. Sample quality

In this set of experiments, we compare sample accuracy and precision among ASRS, RS, and BSRS. Given that ASRS is meant to support both the case of using a sample to obtain the estimate of the entire data stream and the case of using a sample to obtain the estimates of individual sub-streams, experiments are done to report the results in both cases. We refer to the former case as the whole-sample case and the latter case as the sub-sample case.[4] In the sub-sample case, the results are reported as the *average square* value of the sample quality metric used. The results of the experiments demonstrate that in both cases ASRS outperforms both RS and BSRS in sample accuracy as well as precision.

WHOLE-SAMPLE case

Fig. 7(a) and (b) show the ASRS accuracy against the RS accuracy, and Fig. 7(c) and (d) show the ASRS accuracy against the BSRS accuracy, using the synthetic datasets for different degrees of heterogeneity and for different number of sub-streams. Fig. 7(a) and (c) show that the degree of heterogeneity has a major influence on the sample accuracy. For a low degree of heterogeneity (e.g., 10%), we observe that there is only a minor improvement of ASRS accuracy over both RS accuracy and BSRS accuracy. The degree of improvement, however, increases as the degree of heterogeneity increases. For a high degree of heterogeneity (e.g., 70% or higher), we see that the ASRS accuracy is higher than the RS accuracy by more than an order of magnitude. The reason for this is that RS does not consider any heterogeneity between sub-streams whereas ASRS does. For the same high degree of heterogeneity, Fig. 7(c) shows that ASRS achieves an accuracy that is nearly as twice as that achieved by BSRS. These results, which demonstrate that BSRS achieves better accuracy than RS, is reasonable since the basic stratified sampling with proportional allocation is expected to be more appropriate than simple random sampling when the data stream is heterogenous and when the sample is used for the entire data stream. On the other hand, Fig. 7(b) and (d) show that the performance improvement of ASRS over RS and BSRS is more or less constant regardless of the number of sub-streams. This makes sense because the accuracy of ASRS is primarily influenced by the heterogeneity of the data stream (in terms of the size and the values of sub-streams), rather than the actual number of sub-streams.

Fig. 8(a)–(d) show similar results for the sample precision by demonstrating that the degree of heterogeneity has dominant effect on the precision.

[4] In the experiments, q is assigned the values of 1 and 0 for the WHOLE-SAMPLE case and the SUB-SAMPLE case, respectively. (Recall Section 4.2.)

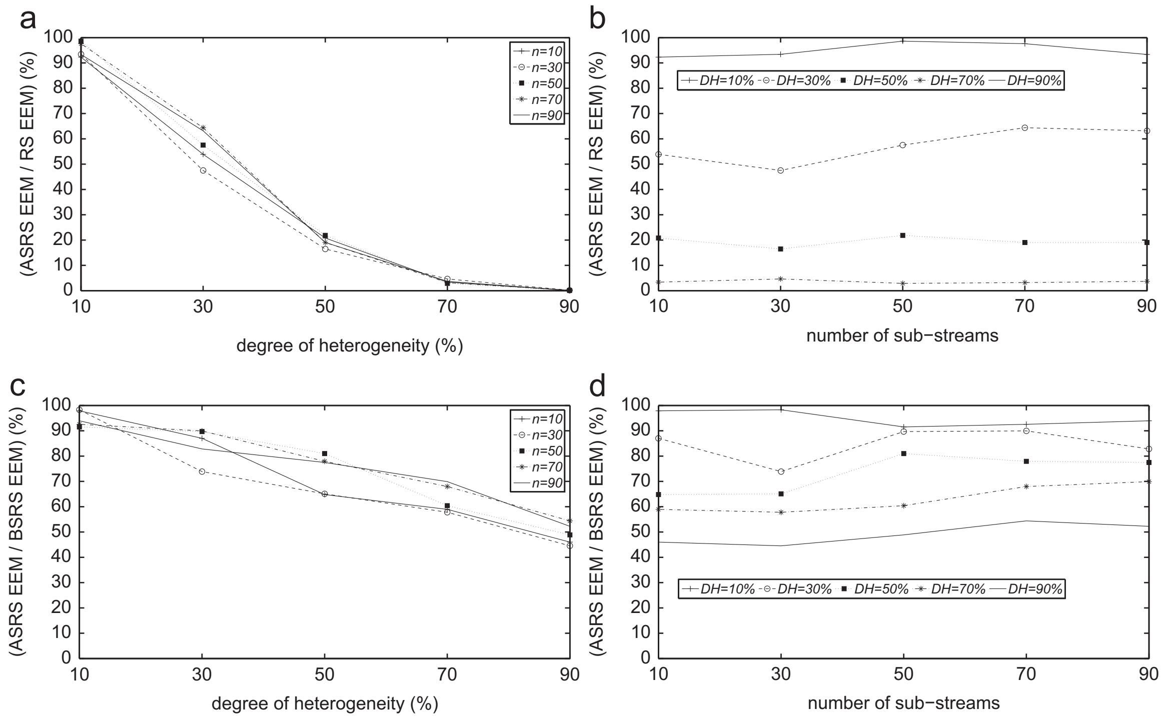

Fig. 7. Whole-Sample accuracy – synthetic datasets. (a) ASRS vs RS: effect of the degree of heterogeneity. (b) ASRS vs RS: effect of the number of sub-streams. (c) ASRS vs BSRS: effect of the degree of heterogeneity and (d) ASRS vs BSRS: effect of the number of sub-streams.

Fig. 9 shows the results from using the real datasets AUCT and SENS. The results are consistent with the results from using the synthetic datasets. The figure shows that the improvement of ASRS over RS and BSRS is higher for the AUCT dataset than SENS with regard to both sample accuracy and sample precision. This is due to the higher degree of heterogeneity of the AUCT dataset.

Sub-sample case

Figs. 10 and 11 show the results for sub-sample accuracy and sub-sample precision, respectively, using the synthetic dataset. These results report the average square value of EEM (for accuracy) and SE (for precision) per sub-sample. As we see in Fig. 10(a) and (c), the sub-sample accuracy of ASRS improves over the accuracy of RS and BSRS linearly with the degree of heterogeneity. Likewise, Fig. 11(a) and (c) show a similar trend for the sub-sample precision. We also observe from Figs. 10(b), (d) and 11(b), (d) that the number of sub-streams is irrelevant to the performance of ASRS, RS, and BSRS at the level of individual sub-samples.

The results from using the SENS and AUCT real datasets in Fig. 12 are similar to those in Figs. 10 and 11.

5.2.2. ASRS adaptivity

In this set of experiments, we demonstrate the adaptivity of the ASRS by showing the change in the allocation of a stratified reservoir sample as a new sub-stream appears in, or an exiting sub-stream expires from, the input stream (i.e., with respect to *data stream membership*) and as the statistical properties of individual sub-streams change over time (i.e., with respect to *sub-streams' stationariness*). Results presented in this section show the change in sub-reservoir sizes over time for five sub-streams synthetically generated and for five sub-streams selected from the AUCT and SENS real datasets. (Only five sub-streams are used for better visibility. Results for a larger number of sub-streams look similar.)

Fig. 13 shows the adaptivity of ASRS from using synthetic datasets. When *DH* is low (10%) (Fig. 13(a)), the sub-reservoir sizes for the sub-streams are relatively close to one another compared with the case of a higher *DH* (90%) (Fig. 13(b)). The observed influence of the *DH* on the closeness of the sub-reservoir sizes is reasonable since the allocation of sub-reservoir size is subject to the heterogeneity of the sub-streams.

Fig. 13(a) and (b) also show that the sizes of sub-reservoirs change more frequently in the early stages of sampling and less frequently as the sampling progresses. The frequent change in the early stages is attributed to the significance of the difference in the sub-streams running statistics. As the sampling progresses, the change in a sub-stream statistics relative to the changes in the statistics of other sub-streams becomes smaller and, therefore, does not cause so much frequent changes in sub-reservoir sizes. This trend is in part due to the fact that the underlying sub-streams are stationary in their statistical properties.

In order to conduct experiments to study the influence of data stream membership and non-stationariness, we modify the generation of synthetic datasets as follows.

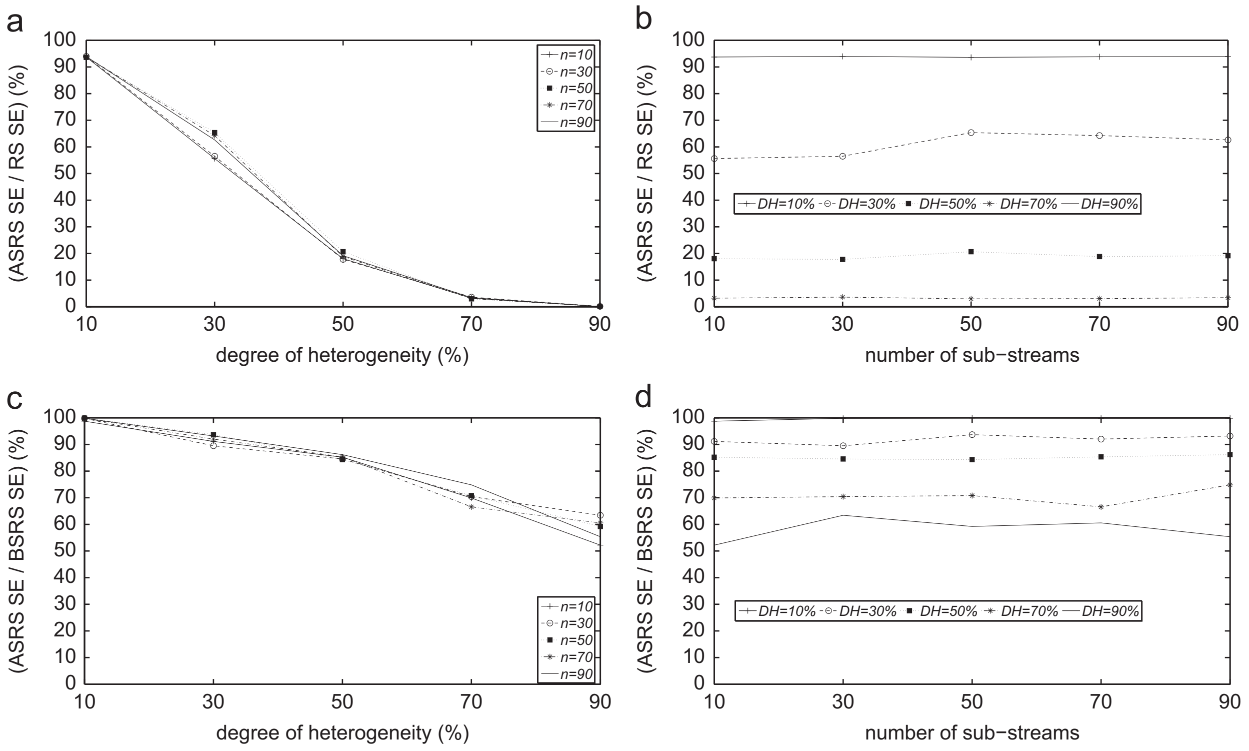

Fig. 8. Whole-Sample precision – synthetic datasets. (a) ASRS vs RS: effect of the degree of heterogeneity. (b) ASRS vs RS: effect of the number of sub-streams. (c) ASRS vs BSRS: effect of the degree of heterogeneity and (d) ASRS vs BSRS: effect of the number of sub-streams.

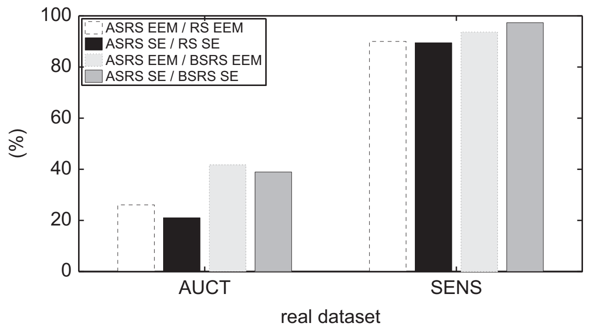

Fig. 9. Whole-Sample accuracy and precision – real datasets.

For data stream membership, we make the sub-streams appear in sequence. For non-stationariness, we periodically re-generate n random numbers that correspond to the *CVs* of n sub-streams such that the overall *DH* among them is preserved (recall Section 5.1.2).

Fig. 13(c) shows that when a new sub-stream appears in a data stream, the ASRS adapts to this situation by releasing memory from the sub-reservoirs of existing sub-streams and allocating the released memory to the sub-reservoirs of the new sub-stream. Fig. 13(d) shows that when the running statistics of some sub-streams change over time, ASRS decreases (or increases) the sizes of some exiting sub-reservoirs and increases (or decreases) the sizes of other exiting sub-reservoirs. A reduced sub-reservoir size may increase afterwards, and vice versa. The frequency of the change in sub-reservoir sizes is relative to the frequency of the change in the running statistics of sub-streams. (Fig. 13(e) shows the case of more frequent change in the running statistics compared to Fig. 13(d).)

Fig. 14(a) shows the change of sub-reservoir sizes using SENS real dataset. This dataset represents the case in which sub-streams all exist from the beginning of the input stream and their statistics remain stationary over time. In other words, readings from different sensors scattered to collect temperature information in a certain area are likely to be generated *altogether* from the time the data collection begins. Besides, the change of temperature readings is expected to be similar at any time of the day. Consequently, all sub-reservoir sizes show little change over time.

Fig. 14(b) shows the change of sub-reservoir sizes using AUCT real dataset. This dataset represents the case in which sub-streams are added one after another and their statistics change over time. Indeed, in auctions applications, it is unlikely that all auctions (represented by sub-streams) start simultaneously; they are expected to start one after another. Besides, the bidding amount of an auction item naturally increases over time, making the statistics of an auction sub-stream non-stationary. As a consequence, we see significant changes of sub-reservoir sizes over time.

Fig. 14(c) further shows the adaptivity of ASRS under the scenario of auctions going open and then closed while sampling progresses. When a new auction opens, memory has to be released from existing sub-reservoirs and

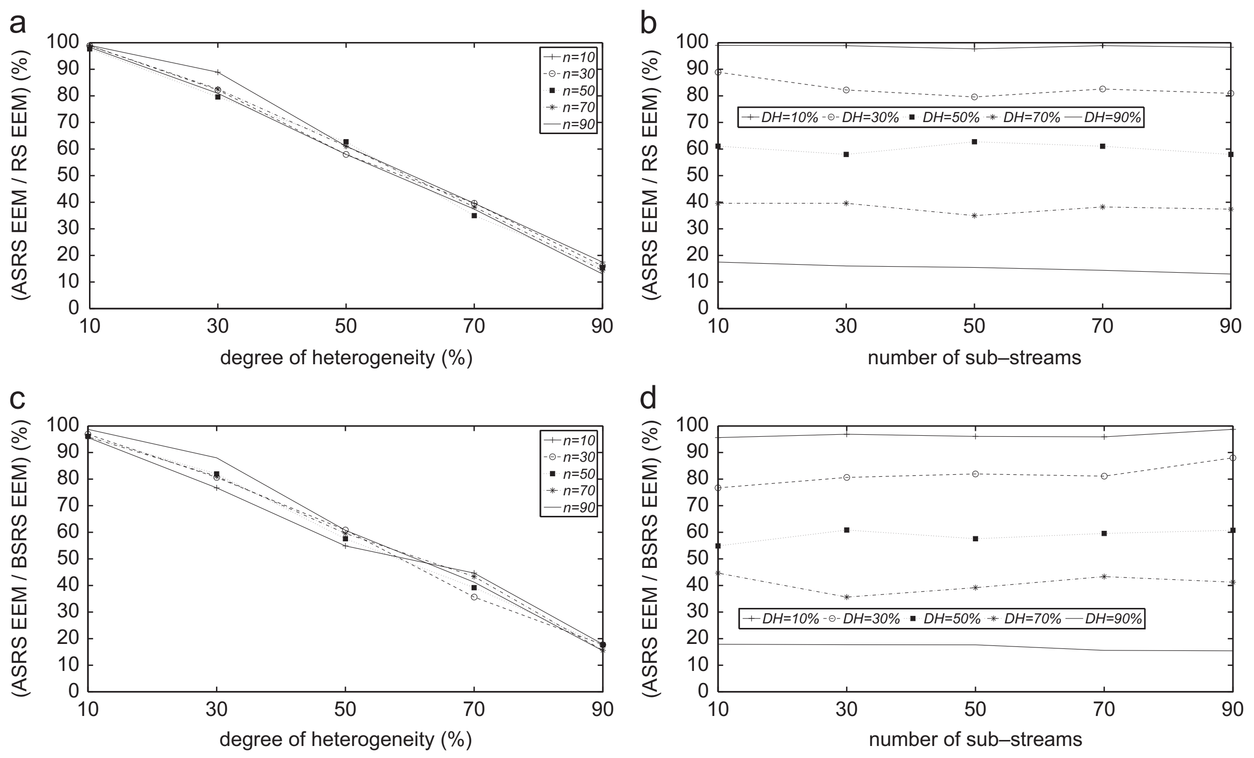

Fig. 10. Sub-Sample accuracy – synthetic datasets. (a) ASRS vs RS: effect of the degree of heterogeneity. (b) ASRS vs RS: effect of the number of sub-streams. (c) ASRS vs BSRS: effect of the degree of heterogeneity and (d) ASRS vs BSRS: effect of the number of sub-streams.

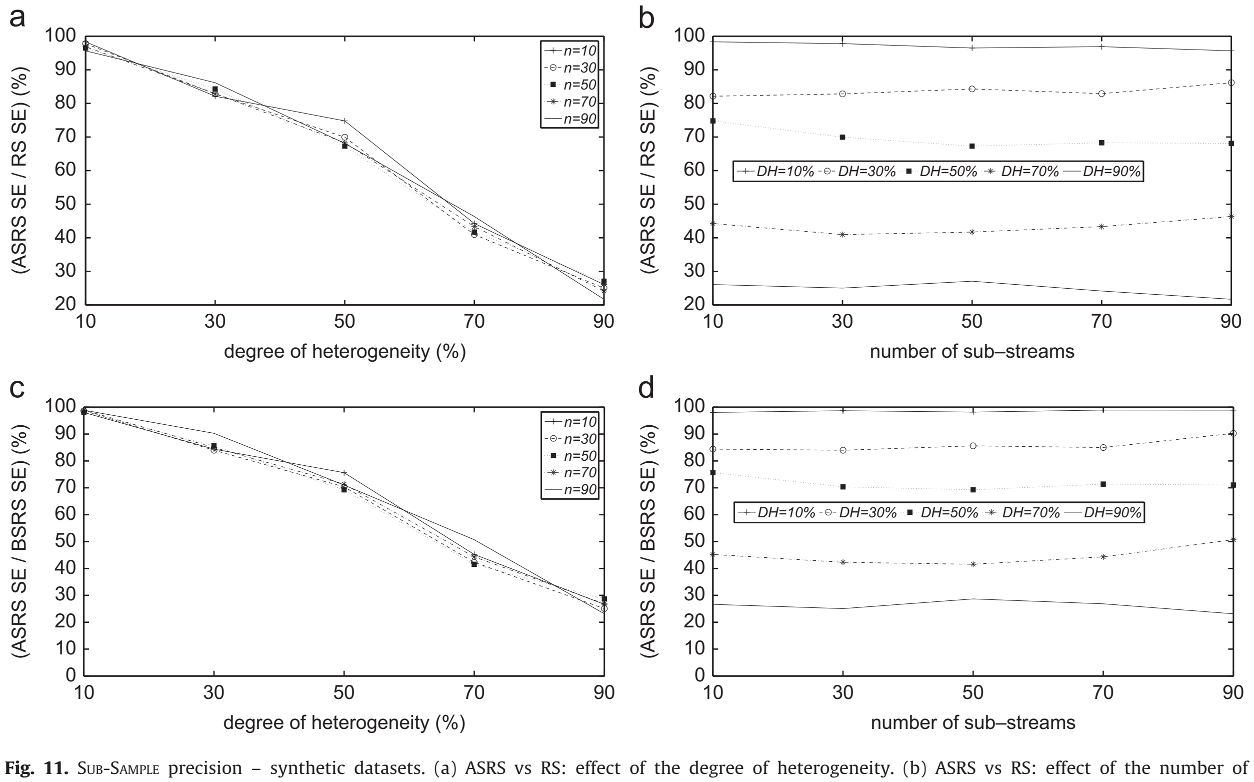

Fig. 11. Sub-Sample precision – synthetic datasets. (a) ASRS vs RS: effect of the degree of heterogeneity. (b) ASRS vs RS: effect of the number of sub-streams. (c) ASRS vs BSRS: effect of the degree of heterogeneity and (d) ASRS vs BSRS: effect of the number of sub-streams.

allocated to the sub-reservoir of the newly opened auction sub-stream (see the point marked with ⁎). When an auction closes from further bids (because the auction is forced to close, the auction expires, the auction item is sold, etc.), the sub-reservoir size of the closed auction sub-stream is released and allocated to the sub-reservoirs of the sub-streams of auctions still open (see the point marked with +).

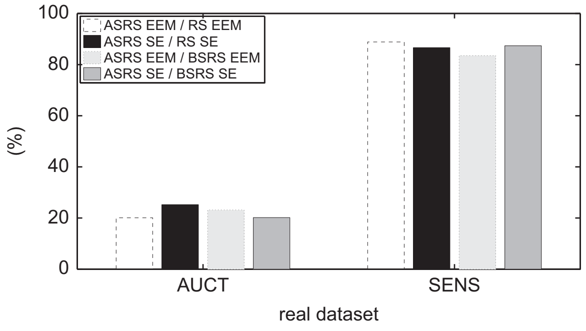

Fig. 12. Sub-Sample accuracy and precision – real datasets.

6. Related work

Existing research on data streams processing includes systems architectures (e.g., [37]), data models (e.g., [38]), query processing (e.g., [39]), data mining (e.g., [40]), data warehousing (e.g., [41]), etc. Our work in this paper pertains to data streams sampling. It is directly related to two main sampling stratified sampling and reservoir sampling, and is generally related to sampling algorithms over data streams.

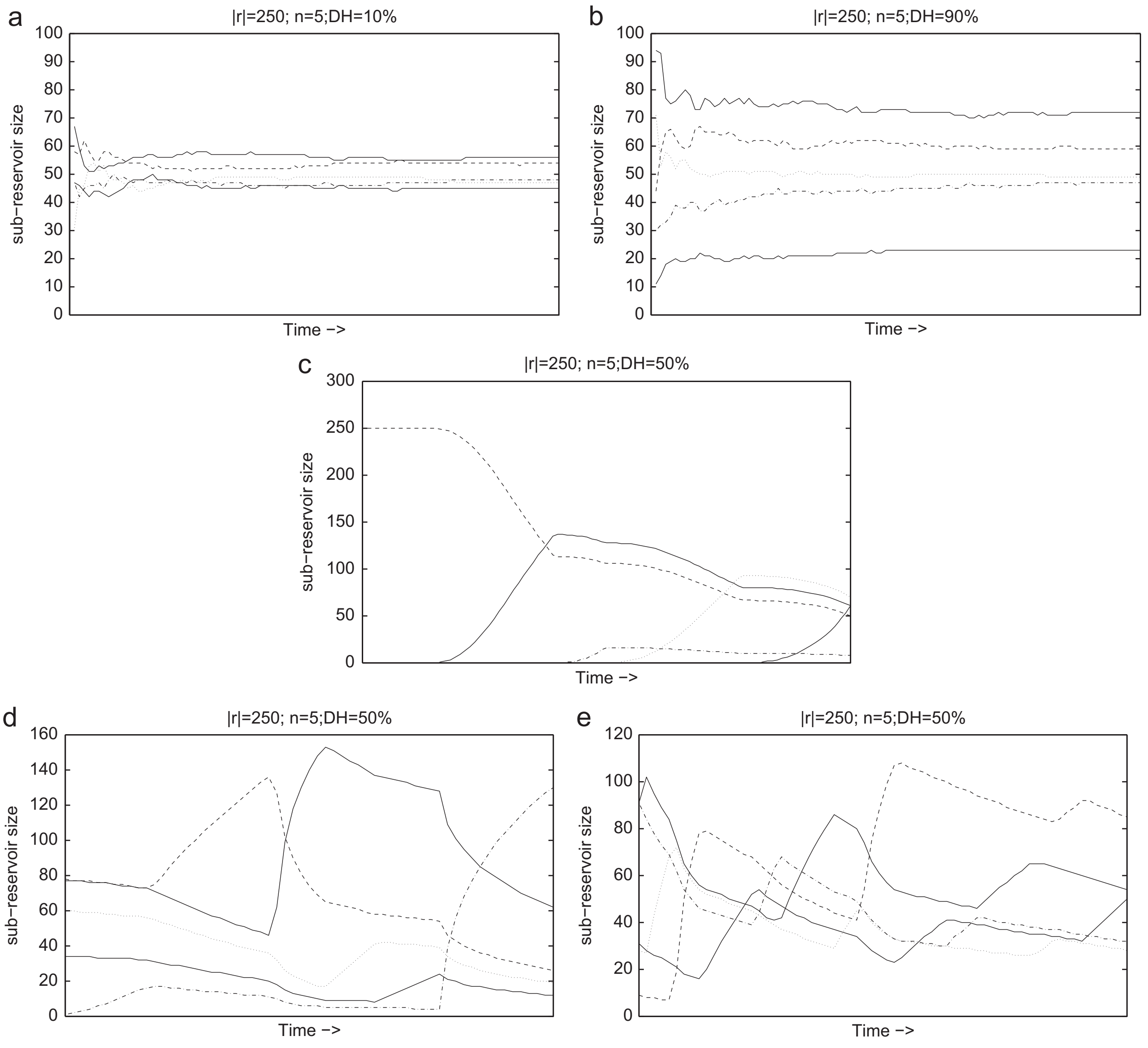

Fig. 13. ASRS adaptivity – synthetic datasets. (a) Low degree of heterogeneity. (b) High degree of heterogeneity. (c) Stream membership. (d) Stream non-stationariness: less frequent change in running statistics and (e) Stream non-stationariness: more frequent change in running statistics.

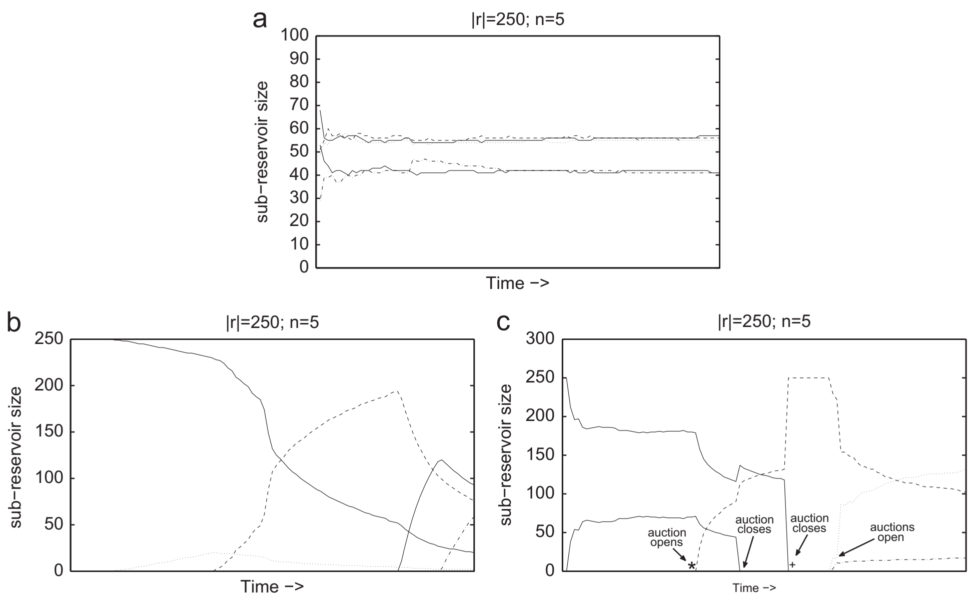

Fig. 14. ASRS adaptivity – real datasets. (a) SENS. (b) AUCT and (c) AUCT: auctions open and close.

6.1. *Stratified sampling*

Stratified sampling has been used for approximate query processing in database systems [42,25,27,28]. *Congressional sampling* [42] proposes a stratified sampling approach to solve the problem of providing accurate approximate answers of a set of grouped aggregation queries using pre-computed biased samples of the data. The idea is to employ stratified sampling to obtain a biased sample in which each stratum corresponds to one of the groups in a given grouped aggregation query. In [25], stratified sampling is used in the problem of identifying a statistically appropriate sample selection for approximate aggregation queries answering. The utility of stratified sampling contributes to the objective of minimizing the overall error in the query result under a given query workload. A comprehensive study of the work proposed in [25] is presented in [27]. The work in [28] solves the problem of using stratified sampling to calculate approximate results of low selectivity aggregation queries. It specifically proposes a Bayesian-based stratified sampling technique that takes the underlying data characteristics into consideration for a better sampling outcome.

All this work on stratified sampling mainly pertains to databases, which makes our work in this paper different as it addresses stratified sampling over *data streams*.

6.2. *Reservoir sampling*

Reservoir sampling technique was proposed by McLeod [17] and has been used in many database applications including clustering [19], data warehousing [22], spatial data management [23], and approximate query processing [12]. Vitter [18] improves the algorithm's performance through more optimization studies, thus achieving the sampling in optimum time with a constant space complexity. The basic idea of the Vitter's algorithm is to repeatedly generate a random number as a function of the reservoir size and the number of tuples seen so far and use the generated number to select the next tuple for inclusion in the reservoir.

Besides the conventional reservoir algorithm, various reservoir-based sampling algorithms have been proposed in the research literature for various applications. Aggarwal [43] proposes a mechanism for biased reservoir sampling over data streams to bias the sample over time using a given temporal bias function. The objective of biased reservoir sampling is to give higher chance for recent data over older data to be included in the reservoir. Gemulla et al. [44,45] address the problem of sampling from an evolving dataset in the presence of insertions and deletions for transactional databases. Specifically, they propose an algorithm for resizing a bounded-size sample assuming that tuples (i.e., transactions) are archived in the system for possible further accesses. In [30], Al-Kateb et al. conduct a theoretical study on the influence of adjusting the size of reservoir in the middle of sampling. Based on these results, they propose a mechanism for maintaining the reservoir sample after the reservoir size is adjusted such that the confidence in the sampling resulting in a uniform sample maintained in the reservoir exceeds a given threshold.

While the conventional reservoir algorithm and its aforementioned variations assume sampling without replacement, other work address the problem of reservoir sampling with replacement in order to allow duplicates in the sample. Park et al. [46] introduce a reservoir sampling algorithm for maintaining a random sample with replacement and show that the performance of the algorithm can be improved by skipping a probabilistically-chosen number of tuples in a fashion similar to Vitter's mechanism [18]. Along the same line, Efraimidis [47] presented reservoir-based algorithms for various weighted random sampling schemes over data streams. Efraimidis examined how the proposed algorithms can handle basic weighted random sampling either with replacement or without replacement and how the algorithms can be further extended to handle the situation in which there is an upper bound on the number of times an item can be replaced.

In contrast to the existing research on reservoir sampling, our work addresses the problem of *stratifying* a reservoir sample rather than maintaining a single reservoir sample.

6.3. *Sampling algorithms over data streams*

Besides reservoir sampling and stratified sampling, there are other stream sampling algorithms proposed in the research literature either to generate a random sample for a specific stream application or for estimating specific aggregates over data streams. Srivastava et al. [15] introduce a random sampling algorithm, known as UNIFORM algorithm, to stream out a uniform random sample over stream joins with limited memory. The UNIFORM algorithm works with two prediction models that provide the number of join-probe tuples produced by each stream tuple. Given this number, the algorithm decides probabilistically if and when a tuple will participate in the sampled join result. In [48], Babcock et al. present two memory-efficient algorithms for the problem of maintaining a uniform random sample of a specified size from a moving window over a data stream. The first algorithm, referred to as chain-sample algorithm, is for sampling from a tuple-based window, and the second algorithm, referred to as priority-sample algorithm, is for sampling from a time-based window. In a recent study, Cohen et al. [49] propose a stream sampling framework for answering subset and rang queries over data streams. The key idea is build up and utilize knowledge of the structure of the data to maintain structure-aware samples. Such samples can then be used to efficiently obtain unbiased estimates for answering arbitrary subset queries, while achieving higher accuracy on range-sum queries than sampling methods that lacks structure-awareness.

Other stream sampling algorithms for estimating specific aggregates include algorithms for distinct values sampling and approximate frequency count. In [50], Gibbons introduces an algorithm for estimating the number of distinct values of a certain attribute (or a group of attributes) over an input stream. Specifically, Gibbons proposes a single-scan sampling algorithm which provides reliable estimates for distinct values queries. The work on approximate frequency counts over data streams [51] deals with the problem of finding tuples that appear in a data stream with a minimum frequency. The proposed algorithm guarantees that tuples included in the output are those that appear in a data stream with a frequency that exceeds a user-given threshold and that the error in estimated frequency does not exceed another user-given threshold.

In contrast to the aforementioned algorithms for sampling over data streams, Johnson et al. [52] abstract the process of sampling from a stream by defining the semantics of a generic stream sampling operator. They examine how such an operator can be instantiated to implement various stream sampling algorithms including distinct values sampling and approximate frequency count outlined above, and demonstrate how it can be implemented efficiently in a real data stream management system.

7. Conclusion

In this paper, we studied the problem of maintaining a stratified sample over data streams which consist of multiple sub-streams with large statistical variations. First, we discussed the motivation of this new research problem in real-world applications. Second, we discussed an optimal allocation method of a fixed-size reservoir, which can be used whether the sample is needed to generate estimates of the whole data stream or the sub-streams on an individual basis. Third, we presented a sampling algorithm which uses the proposed allocation method to adjust the allocation of a stratified reservoir sample among sub-streams adaptively as sub-streams appear in, or disappear from, the input stream and as their statistical properties change over time. Finally, through experiments, we demonstrated the adaptivity of the proposed algorithm and its superiority over the conventional reservoir sampling algorithm with regard to the sample quality.

Several issues are open for future work. One issue is to extend the proposed algorithm to handle *multi-variate* sampling situation in which an input stream has multiple sampling attributes and an estimate is needed from each sampling attribute. In this situation, it may be required to compromise the allocation of a stratified reservoir sample with respect to the target estimates. Another issue is to explore the utility of the proposed algorithm in more real-world applications.

Acknowledgments

The authors would like to thank the members of Intel Berkeley Research lab for graciously granting the permission to use their sensor datasets in the experiments. The authors would also like to thank the Federal Communications Commission for making their auctions data available to use in the experiments. This material is based upon work supported by the National Science Foundation under Grant No. IIS-0415023.

Appendix A. Calculating a running standard deviation

A straightforward way to compute the standard deviation is the standard two-pass method [33], which requires two passes over the underlying data – once to compute the mean and once to compute the variance. This method is not usable for data streams because it requires all data values to be stored.

Naturally, we need a way to compute a *running* standard deviation in a single pass as tuples are arriving over data stream. The *textbook one-pass* method [33] does it by keeping track of two values, that is, the sum of squares and the square of sum of the variable values

($\sum y_{ij}^2$ and $(\sum y_{ij})^2$) over incoming data values. That is,

$$\sigma_i(t) = \sqrt{\frac{1}{|S_i(t)|}\left(|S_i(t)|\sum_{j=1}^{|S_i(t)|} y_{ij}^2 - \left(\sum_{j=1}^{|S_i(t)|} y_{ij}\right)^2\right)} \quad \text{(A.1)}$$

This method, however, has a major drawback. If y_{ij}s have large values but small differences, then the calculation may generate very small numbers as a result of subtracting between two large numbers. This may cause a loss of numerical precision to the extent of the subtraction producing a negative number [33]. Thus, we use another method, known as the *updating* method [33], which features both the single-pass and the precise computation. This method employs a recurrence relation that is capable of calculating the standard deviation in a single scan of the data and providing precise calculation even when the data values are large. The recurrence relation of this method is as follows:

$$W_i = W_{i-1} + \frac{1}{i}(y_i - W_{i-1}) \quad \text{(A.2)}$$

$$Z_i = Z_{i-1} + (i-1)(y_i - W_{i-1})\left(\frac{y_i - W_{i-1}}{i}\right) \quad \text{(A.3)}$$

where $W_1 = y_1$ and $Z_1 = 0$.

Appendix B. Adjusting the reservoir size

B.1. Decreasing the reservoir size

Suppose the size of a reservoir is decreased from $|r|$ to $|r|-\delta$ $(\delta > 0)$ immediately after the kth tuple arrives. Then, the sample in the reduced reservoir can be maintained by randomly evicting δ tuples from the original reservoir. With this random eviction in place, there are $\binom{|r|}{|r|-\delta}$ samples that can be selected in the reduced reservoir from the original reservoir.

Note that there are $\binom{k}{|r|}$ different samples of size $|r|$ that can be selected in the original reservoir from the k tuples and there are $\binom{k-(|r|-\delta)}{|r|-(|r|-\delta)}$ duplicate samples of size $|r|-\delta$ that can be selected in the reduced reservoir from the different samples of size $|r|$. Therefore, there are $(\binom{k}{|r|}\binom{|r|}{|r|-\delta})/\binom{k-(|r|-\delta)}{|r|-(|r|-\delta)}$ different samples of size $|r|-\delta$ that can be selected in the reduced reservoir from the k tuples. On the other hand, the number of different samples of size $|r|-\delta$ that should be statistically possible from sampling k tuples is $\binom{k}{|r|-\delta}$. Hence, the uniformity confidence is expressed as follows:

$$UC(k,|r|,\delta) = \frac{\binom{k}{|r|}\binom{|r|}{|r|-\delta}/\binom{k-(|r|-\delta)}{|r|-(|r|-\delta)}}{\binom{k}{|r|-\delta}}100 = \frac{\binom{k}{|r|-\delta}}{\binom{k}{|r|-\delta}}100 \quad \text{(B.1)}$$

which clearly shows that the uniformity confidence is 100%.

The following theorem summarizes the uniformity confidence property of reservoir sampling in the event of decreasing the reservoir size during sampling.

Theorem 1. *If the size of a reservoir is decreased from* $|r|$ *to* $|r|-\delta$ $(\delta > 0)$ *while sampling from an input stream is in progress, it is possible to maintain the sample in the reduced reservoir with a* 100% *uniformity confidence.*

Proof. To prove that the uniformity confidence is 100%, we only need to show that every tuple among the k tuples seen so far has an equal probability to be selected in the reduced reservoir. Following the conventional reservoir sampling, each of the k tuples has the equal probability $|r|/k$ to be selected in the original reservoir. Once the reservoir size decreases to $|r|-\delta$, we can sample the tuples in the original reservoir with the equal probability $(|r|-\delta)/|r|$ to select $|r|$-δ tuples for the reduced reservoir. Thus, every tuple among the k tuples has the equal probability

$$\frac{|r|}{k} \times \frac{|r|-\delta}{|r|} = \frac{|r|-\delta}{k}$$

to be selected in the reduced reservoir. □

B.2. Increasing the reservoir size

Suppose the size of a reservoir is increased from $|r|$ to $|r|+\delta$ $(\delta > 0)$ immediately after the kth tuple arrives. Then, the reservoir has room for δ additional tuples. Clearly, there is no way to fill this room from sampling the k tuples as they have already passed by. We can only use incoming tuples to fill the room. Hence, the uniformity confidence is definitely less than 100%.

The following theorem summarizes the uniformity confidence property of reservoir sampling in the event of increasing the reservoir size during sampling.

Theorem 2. *If the size of a reservoir is increased from* $|r|$ *to* $|r|+\delta$ $(\delta > 0)$ *while sampling from an input stream is in progress (after seeing more than* $|r|$ *tuples), it is not possible to maintain the sample in the enlarged reservoir with a* 100% *uniformity confidence.*

Proof. Let i be the number of tuples that can be selected in the enlarged reservoir (of size $|r|+\delta$) from the k tuples seen so far in the input stream. Then, the uniformity confidence is equal to 100% if and only if i can be any value in the range of [0, $|r|+\delta$]. However, i cannot be more than $|r|$ since we have only $|r|$ tuples from the k tuples seen so far. From this we conclude that the uniformity confidence cannot reach 100%. □

References

[1] W.G. Cochran, Sampling Techniques, third ed., John Wiley, 1977.
[2] K. Clarkson, A probabilistic algorithm for the post office problem, in: Proceedings of the 17th Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, 1985, pp. 175–184.
[3] K.L. Clarkson, Applications of random sampling in computational geometry, II, in: Proceedings of the 4th Annual Symposium on Computational Geometry, ACM, New York, NY, USA, 1988, pp. 1–11.
[4] D.R. Karger, Random Sampling in Graph Optimization Problems, Ph.D. thesis, Stanford University, Stanford, CA, USA, 1995.
[5] J. Kivinen, H. Mannila, The power of sampling in knowledge discovery, in: Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM, New York, NY, USA, 1994, pp. 77–85.

[6] S.D. Lee, D.W. Cheung, B. Kao, Is sampling useful in data mining? a case in the maintenance of discovered association rules, Data Mining and Knowledge Discovery 2 (1998) 233–262.

[7] S. Acharya, P.B. Gibbons, V. Poosala, S. Ramaswamy, Join synopses for approximate query answering, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 1999, pp. 275–286.

[8] S. Chaudhuri, R. Motwani, V. Narasayya, On random sampling over joins, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 1999, pp. 263–274.

[9] P.B. Gibbons, Y. Matias, New sampling-based summary statistics for improving approximate query answers, in: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 1998, pp. 331–342.

[10] P.B. Gibbons, Y. Matias, V. Poosala, Fast incremental maintenance of approximate histograms, in: Proceedings of the 23rd International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, pp. 466–475.

[11] F. Olken, Random Sampling from Databases, Ph.D. thesis, University of California at Berkeley, Berkeley, California, USA, 1993.

[12] D. Yi-Leh Wu, Agrawal, A. El Abbadi, Query estimation by adaptive sampling, in: Proceedings of the 18th International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA, 2002, pp. 639–648.

[13] B. Babcock, M. Datar, R. Motwani, Load shedding for aggregation queries over data streams, in: Proceedings of the 20th International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA, 2004, pp. 350–361.

[14] C. Jermaine, A. Pol, S. Arumugam, Online maintenance of very large random samples, in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2004, pp. 299–310.

[15] U. Srivastava, J. Widom, Memory-limited execution of windowed stream joins, in: Proceedings of the 30th International Conference on Very Large Data Bases, VLDB Endowment, 2004, pp. 324–335.

[16] G. Cormode, S. Muthukrishnan, K. Yi, Q. Zhang, Optimal sampling from distributed streams, in: Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '10, ACM, New York, NY, USA, 2010, pp. 77–86.

[17] A. McLeod, D. Bellhouse, A convenient algorithm for drawing a simple random sample, Applied Statistics 32 (1983) 182–184.

[18] J.S. Vitter, Random sampling with a reservoir, ACM Transactions on Mathematical Software 11 (1) (1985) 37–57.

[19] S. Guha, R. Rastogi, K. Shim, CURE: an efficient clustering algorithm for large databases, in: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 1998, pp. 73–84.

[20] K. Kerdprasop, N. Kerdprasop, P. Sattayatham, Density-biased clustering based on reservoir sampling, in: DEXA Workshops '05, 2005, pp. 1122–1126.

[21] K. Kerdprasop, N. Kerdprasop, J. Sun, Density biased reservoir sampling for clustering, in: Artificial Intelligence and Applications, 2005, pp. 95–100.

[22] P.G. Brown, P. J. Haas, Techniques for warehousing of sample data, in: Proceedings of the 22nd International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA, 2006, pp. 6–17.

[23] F. Olken, D. Rotem, Sampling from spatial databases, in: Proceedings of the 9th International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA, 1993, pp. 199–208.

[24] P.B. Gibbons, Y. Matias, V. Poosala, Fast incremental maintenance of approximate histograms, ACM Transactions on Database Systems 27 (2002) 261–298.

[25] S. Chaudhuri, G. Das, V. Narasayya, A robust, optimization-based approach for approximate answering of aggregate queries, in: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2001, pp. 295–306.

[26] V. Ganti, M.-L. Lee, R. Ramakrishnan, ICICLES: Self-tuning samples for approximate query answering, in: Proceedings of 26th International Conference on Very Large Data Bases, September 10–14, 2000, Cairo, Egypt, Morgan Kaufmann, 2000.

[27] S. Chaudhuri, G. Das, V. Narasayya, Optimized stratified sampling for approximate query processing, ACM Transactions on Database Systems 32 (2) (2007) 9.

[28] S. Joshi, C.M. Jermaine, Robust stratified sampling plans for low selectivity queries, in: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA, 2008, pp. 199–208.

[29] M.D. Bankier, Power allocations: determining sample sizes for subnational areas, The American Statistician 42 (1988) 174–177.

[30] M. Al-Kateb, B. S. Lee, X. S. Wang, Adaptive-size reservoir sampling over data streams, in: Proceedings of the 19th International Conference on Scientific and Statistical Database Management, IEEE Computer Society, Banff, Canada, 2007, pp. 22–33.

[31] FCC, FCC Auctions 〈http://wireless.fcc.gov/auctions/default.htm〉, 2005.

[32] J. Neyman, On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection, Journal of the Royal Statistical Society 97 (1934) 558–625.

[33] R.J.L. Tony, F. Chan, G.H. Golub, Algorithms for computing the sample variance: analysis and recommendations, The American Statistician 37 (1983) 242–247.

[34] J.K. Patel, C.B. Read, Handbook of the Normal Distribution, CRC, 1996.

[35] R. Norgaard, T. Killeen, Expected utility and the truncated normal distribution, Management Science 26 (1980) 901–909.

[36] Intel lab data 〈http://berkeley.intel-research.net/labdata/〉.

[37] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, J. Widom, STREAM: the Stanford stream data manager, IEEE Data Engineering Bulletin 26 (1) (2003) 19–26.

[38] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, 2002, pp. 1–16.

[39] A. Das, J. Gehrke, M. Riedewald, Approximate join processing over data streams, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2003, pp. 40–51.

[40] J. Guo, P. Zhang, J. Tan, L. Guo, Mining frequent patterns across multiple data streams, in: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11, ACM, New York, NY, USA, 2011, pp. 2325–2328.

[41] A. Cuzzocrea, Data warehousing and knowledge discovery from sensors and streams, Knowledge and Information Systems 28 (3) (2011) 491–493.

[42] S. Acharya, P. B. Gibbons, V. Poosala, Congressional samples for approximate answering of group-by queries, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2000, pp. 487–498.

[43] C.C. Aggarwal, On biased reservoir sampling in the presence of stream evolution, in: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12–15, 2006, pp. 607–618.

[44] R. Gemulla, W. Lehner, P.J., Maintaining bounded-size sample synopses of evolving datasets, The VLDB Journal 17(2) (2008) 173–201.

[45] R. Gemulla, W. Lehner, P.J. Haas, A dip in the reservoir: maintaining sample synopses of evolving datasets, in: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12–15, 2006, pp. 595–606.

[46] B.-H. Park, G. Ostrouchov, N. F. Samatova, A. Geist, Reservoir-based random sampling with replacement from data stream, in: Proceedings of the 4th SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22–24, 2004.

[47] P.S. Efraimidis, Weighted random sampling over data streams, CoRR, 2010.

[48] B. Babcock, M. Datar, R. Motwani, Sampling from a moving window over streaming data, in: Proceedings of the 13th Annual ACM-SIAM symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002, pp. 633–634.

[49] E. Cohen, G. Cormode, N. Duffield, Structure-aware sampling on data streams, in: Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '11, ACM, New York, NY, USA, 2011, pp. 197–208.

[50] P.B. Gibbons, Distinct sampling for highly accurate answers to distinct values queries and event reports, in: Proceedings of the 27th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001, pp. 541–550.

[51] G.S. Manku, R. Motwani, Approximate frequency counts over data streams, in: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB '02, VLDB Endowment, 2002, pp. 346–357.

[52] T. Johnson, S. Muthukrishnan, I. Rozenbaum, Sampling algorithms in a stream operator, in: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2005, pp. 1–12.