**The VLDB Journal manuscript No.**
(will be inserted by the editor)

# Non-Uniformity Issues and Workarounds in Bounded-Size Sampling

**Rainer Gemulla · Peter J. Haas · Wolfgang Lehner**

Received: date / Accepted: date

**Abstract** A variety of schemes have been proposed in the literature to speed up query processing and analytics by incrementally maintaining a bounded-size uniform sample from a dataset in the presence of a sequence of insertion, deletion, and update transactions. These algorithms vary according to whether the dataset is an ordinary set or a multiset and whether the transaction sequence consists only of insertions or can include deletions and updates. We report on subtle non-uniformity issues that we found in a number of these prior bounded-size sampling schemes, including some of our own. We provide workarounds that can avoid the non-uniformity problem; these workarounds are easy to implement and incur negligible additional cost. We also consider the impact of non-uniformity in practice and describe simple statistical tests that can help detect non-uniformity in new algorithms.

**Keywords** Database sampling · Reservoir sampling · Bernoulli sampling · Sample maintenance

## 1 Introduction

Use of random samples can speed up database querying and analytics by orders of magnitude. Because such samples are usually too expensive to compute on demand, there has been growing interest in algorithms for incrementally maintaining a random sample as the underlying dataset evolves due to a transaction sequence of insertions, deletions, or updates of data items; see [4, 10, 12, 13, 15] and references therein. To keep costs low, these algorithms avoid expensive accesses to the dataset itself, and touch only the sample and the transaction sequence. Most of the algorithms maintain a uniform random sample (defined below), because such samples are extremely flexible in applications and are the focus of a large body of statistical estimation theory.[1] Maintaining bounded-size samples is also desirable, because such samples simplify memory management and allow control of the computational costs for algorithms that use the samples. In this note, we describe some rather subtle non-uniformity issues that we found in earlier algorithms (including ours) for incrementally maintaining bounded-size uniform samples. Departures from uniformity in a sample-maintenance scheme can lead to errors in sample-based statistical procedures, such as point or interval estimation, model fitting, or hypothesis testing.

Denote by $R$ a dataset of interest and let $S \subseteq R$ be a sample of $R$. We focus on the case where $R$ is not a static dataset but evolves over time. That is, dataset $R$ is subject to a sequence of update, deletion, and insertion (UDI) transactions—in the following, we focus on insertions and deletions, since an update can be viewed as a deletion followed by an insertion. The job of a sample-maintenance algorithm is to keep the sample $S$ "in sync" with the evolving dataset. In our setting, this means that $S$ should constitute a truly uniform random sample of $R$ after each transaction has been processed. We consider sample-maintenance algorithms that are

R. Gemulla
Max Planck Institute for Informatics, Saarbrücken, Germany
E-mail: rgemulla@mpi-inf.mpg.de

P. J. Haas
IBM Almaden Research Center, San Jose, California, USA
E-mail: phaas@us.ibm.com

W. Lehner
Technische Universität Dresden, Germany
E-mail: wolfgang.lehner@tu-dresden.de

[1] There is also a body of literature devoted to efficient maintenance of various kinds of deliberately non-uniform samples, usually with some particular analysis task in mind. See [5] for a recent example, aimed at estimating "subset sums."

designed for the case in which $R$ is subject only to insertions, and for the case in which $R$ is subject to a general UDI sequence. (In the insertion-only setting, our setup also models the task of computing, in a single scan, a sample of a dataset of unknown size.)

Uniformity of samples is meant to capture the intuitive notion of randomness, and its precise definition depends on the type of sampling involved. Specifically, sample-maintenance algorithms can be classified according to whether the dataset $R$ and the sample $S$ are sets or multisets. In the context of *set sampling*, both $R$ and $S$ are sets, and $S$ is said to be *uniform* if $\Pr[S = A] = \Pr[S = B]$ for all pairs $A, B \subseteq R$ satisfying $|A| = |B|$, where $|U|$ denotes the number of elements in set $U$. Subsets of equal size are therefore produced by the sampling algorithm with equal probability. The uniformity condition can be restated concisely as

$$\Pr[S = A] = \Pr[|S| = |A|] \binom{|R|}{|A|}^{-1}$$

for all $A \subseteq R$. In *multiset sampling*, both $R$ and $S$ are multisets, so that duplicate values are allowed, and items having the same value are indistinguishable. For a multiset $A$, denote by $|A|$ the size of $A$ (including duplicates), by $D(A)$ the set of distinct values in $A$, and by $|A(r)|$ the frequency of $r$ in $A$ for $r \in D(A)$. We use the usual multiset semantics and write $S \subseteq R$ if and only if $|S(r)| \leq |R(r)|$ for $r \in D(S)$. Suppose that $R = \{3, 3, 7\}$ and consider a size-1 sample $S$ of $R$. Since the value 3 occurs twice as often as the value 7 in $R$, we want $\Pr[S = \{3\}] = 2\Pr[S = \{7\}]$. In general, a multiset sample $S$ of is *uniform* if

$$\Pr[S = A] = \Pr[|S| = |A|] \binom{|R|}{|A|}^{-1} \prod_{r \in D(A)} \binom{|R(r)|}{|A(r)|} \tag{1}$$

for all $A \subseteq R$. In *distinct-item sampling*, $R$ is a multiset and $S$ is a subset of $D(R)$, so that $S$ is an ordinary set. A multiset sample $S$ of size $n$ is *uniform* if

$$\Pr[S = A] = \Pr[|S| = |A|] \binom{|D(R)|}{|A|}^{-1}$$

for all $A \subseteq D(R)$.

There are two basic sampling schemes that underlie sample maintenance in both set sampling and multiset sampling scenarios: Bernoulli sampling and reservoir sampling; see Sec. 2.1 below for details. Bernoulli samples are easy to implement and parallelize, but offer only probabilistic bounds on the sample size. (The expected sample size is proportional to the dataset size.) Reservoir samples are more involved to implement and parallelize but provide a bounded sample size. Both schemes were originally developed for insertion-only transaction sequences, but have been extended to deal with deletion transactions; see Sec. 3. Several strategies have been proposed to try and profit from both the ease of Bernoulli sampling and the above-mentioned benefits of maintaining strict bounds on the sample size. One approach, used for "growing" datasets in which the long-run average rate of insertions exceeds the long-run average rate of deletions, is to start with a Bernoulli sample and switch over to reservoir sampling as soon as the sample size hits a specified upper bound. Examples of this approach include "hybrid Bernoulli sampling" [4] and "Bernoulli resizing" [12]. The goal of the former method is to use the Bernoulli scheme as long as possible to facilitate parallel sampling, and the goal of the latter method is to use Bernoulli sampling for as short a period as possible to dynamically increase the sample-size bound in the presence of a growing dataset. A related approach that we call "Bernoulli sampling with purging" (see [13]) uses Bernoulli subsampling to reduce the sample size—or sample footprint in the case of multisets—whenever it exceeds a specified bound. In the setting of distinct-item sampling, the "dynamic inverse sampling" method [6] proceeds rather differently than the above methods and makes use of a set of pairwise-independent hash functions.

In Secs. 2–7 we show, perhaps surprisingly, that all of these methods, as well as some "intuitive" additional methods, can produce non-uniform samples. A few of these results have appeared in the literature [4, 12] and we summarize them for completeness, but most of the analyses are new. These non-uniformity issues often arise in rather subtle ways. For example, bounded-size Bernoulli samples do not constitute true Bernoulli samples, and switch-over or subsampling procedures that rely on characteristics of true Bernoulli samples lead to non-uniformity. We provide alternatives to each of the incorrect algorithms, using novel ideas such as randomizing switch-over times, invoking subsampling based on the dataset size rather than the sample size, and replacing Bernoulli sampling with a "random pairing" scheme during sample resizing. In some cases, these workarounds eliminate the non-uniformity problems while still providing strict sample-size bounds; in other cases, one must settle for probabilistic sample-size bounds, though the exceedance probabilities for these bounds are tightly controlled.

Table 1 summarizes our results. In the table, we classify sampling scenarios as set sampling, multiset sampling, and distinct-item sampling. Within each of these categories, we distinguish between insertion-only transaction sequences ("I-only") and UDI sequences.

**Table 1** Overview of bounded-size sampling schemes. U = uniform, NU = non-uniform, and * indicates a new workaround algorithm. Note that a scheme that is uniform for UDI transactions is also uniform for insertion-only transactions; a scheme that is non-uniform for insertion-only transactions is also non-uniform for UDI transactions.

| | | Name | Abbr. | Section | Sample size |
|---|---|---|---|---|---|
| **Set Sampling** | | | | | |
| I-Only | U | Reservoir sampling [7] | RS($M$) | 2 | Bounded |
| | | Bernoulli sampling with purging [13][a] | BSP($q_0, M$) | 2.2 | Bounded |
| | | Hybrid Bernoulli sampling with randomized switch-over* | HBSR($q, M$) | 2.4 | Bounded |
| | NU | Hybrid Bernoulli sampling [4] | HBS($q, M$) | 2.1 | Bounded |
| | | Adapted reservoir sampling [15] | ARS($M$) | 3.1 | Bounded |
| | | Adapted counting sampling [15] | ABSP($M$) | 3.4 | Bounded |
| UDI | U | Bernoulli sampling with probabilistic bounds* | PBS($M, \delta$) | 2.3, 3.5 | Prob. bounded |
| | | Random pairing [12] | RP($M$) | 3.2 | Bounded |
| | | Random pairing resizing* | RPR | 7.2 | Dyn. bounded |
| | NU | Bernoulli sampling with purging [13] | BSP($q_0, M$) | 2.2 | Bounded |
| | | Hybrid Bernoulli sampling with randomized switch-over | HBSR($q, M$) | 3.3 | Bounded |
| | | Bernoulli resizing [12] | BR | 7.1 | Dyn. Bounded |
| **Multiset Sampling** | | | | | |
| I-Only | U | Any insertion-only, bounded[b] set-sampling method | | 4 | Bounded |
| | NU | Bernoulli sampling with purging (bounded footprint) [13] | BSP($q_0, F$) | 3.4 | Bounded footprint |
| UDI | U | Augmented Bernoulli sampling with probabilistic bounds* | PABS($F, \delta$) | 4.2 | Prob. bounded footprint |
| **Distinct-Item Sampling** | | | | | |
| I-Only | U | Min-wise hashing [2] | MIN($M$) | 5.3 | Bounded |
| UDI | U | Distinct-item Bernoulli sampling with. prob. bounds* | DPBS($F, \delta$) | 5.2 | Prob. bounded footprint |
| | | Distinct-item subsampling for PABS($F, \delta$) [11] | PABSD($F, \delta$) | 5.4 | Prob. bounded footprint |
| | | Augmented Min-wise hashing [1] | AKMV($F$) | 5.3 | Bounded footprint |
| | | Dynamic inverse sampling, indep. hashing* | DIS$_\infty$($F$) | 5.1 | Bounded footprint |
| | NU | Dynamic inverse sampling, pairwise-indep. hashing [6] | DIS$_2$($F$) | 5.1 | Bounded footprint |

[a] Correctness is conjectured but not proved. [b] Bounded means bounded sample size, not bounded footprint.

A sampling scheme is classified as "Bounded" if it produces strictly bounded samples (i.e., samples that are bounded with probability 1), as "probabilistically bounded ("Prob. bounded") if the bound may be exceeded, but the exceedance probability is small and tightly controlled, and as dynamically bounded ("Dyn. bounded") if the samples are strictly bounded at any point in time, but the sample-size bound can be increased over time as the dataset grows. Some of these results are of primarily of theoretical interest; in Sec. 9, we give practical guidelines on which methods to use in various scenarios.

In Sec. 8, we discuss several practical issues pertinent to our results. Given the number of incorrect methods that have appeared in the literature, we first describe in Sec. 8.1 a simple statistical procedure that we have found useful in identifying departures from uniformity in purportedly uniform sampling procedures. We then turn to the practical impact of non-uniformity. It is difficult in general to theoretically quantify the magnitude of departures from uniformity, and even harder to quantify the practical effects of non-uniformity on analytic or statistical procedures that assume truly uniform samples. To develop intuition, we provide a numerical case study (Sec. 8.2) of the non-uniformity effects arising from the use of hybrid Bernoulli samples. Our results indicate that the degree of non-uniformity becomes small as the sample size increases. Although a simple example shows that the absolute estimation error for a given degree of non-uniformity is unbounded, the relative error tends to be small. We believe that it would be a mistake, however, to take these results to mean that non-uniformity issues can simply be ignored. As is well known in the context of uniform pseudo-random number generation, even small, subtle departures from uniformity can result in highly anomalous statistical results; see, e.g., [9]. There is every reason to fear that similar problems will arise from the non-uniformity of "uniform" samples over the wide range of applications for which such samples are used. Since the workarounds described here typically are as easy to implement as the incorrect versions and incur no significant additional overheads, there is no reason to use incorrect non-uniform sampling methods.

## 2 Set Sampling for Insertion-Only Sequences

In this section, we assume that the sequence of transactions consists of only insertions and that $R$ is a set. We discuss deletions in Sec. 3, multisets in Secs. 4 and 5, merging problems in Sec. 6, and resizing problems in Sec. 7.

To set the stage for our discussion of bounded-size set-sampling algorithms, we briefly describe both Bernoulli sampling and reservoir sampling in the insertion-only setting. Both schemes start with an initially empty sample. In Bernoulli sampling with sampling rate $q$, abbreviated Bern($q$), each arriving item is accepted into the sample with probability $q$ and rejected with probability $1-q$, independently of the other items. In reservoir sampling with sample size parameter $M$, abbreviated RS($M$), the first $M$ items are directly added to the sample. Subsequent items are accepted with probability $M/i$ and rejected otherwise, where $i$ is the number of items already processed (including the new item). In case of acceptance, the new item replaces a randomly and uniformly selected sample item. Note that the sample size of RS($M$) equals $M$ after processing $i \geq M$ items. In contrast, the sample size of a Bern($q$) sample follows a Binomial($i, q$) distribution; the sample has mean size $iq$ and thus grows with the dataset.

As shown in the following sections, a hybrid scheme that naively switches from Bernoulli to reservoir sampling when the sample size hits an upper bound leads to non-uniformity. A couple of workarounds use Bernoulli sampling only, but with occasional "purges," that is, subsampling steps, to control the sample size. A different workaround retains the switch-over strategy, but randomizes the switch-over time. It is important to note that these workarounds provide random-sized samples that are truly uniform (equal-size samples have equal probability of being produced) but, if the sample size is bounded, then the sample cannot be a true Bernoulli sample even during the "Bernoulli sampling" phase of an algorithm. That is, the sample-size distribution is not binomial as with Bernoulli samples, but is usually rather complex.

### 2.1 Incorrect: Hybrid Bernoulli sampling

Consider a switch-over algorithm as in [4,12]–denoted HBS($q, M$)—that initially uses Bernoulli sampling with rate $q$ and then switches to reservoir sampling as soon as the sample size reaches an upper bound $M > 0$. Suppose that $q = 1/2$ and $M = 2$, and that we process the sequence $+r_1, +r_2, +r_3$ of three insertions into an initially empty dataset. Denote by $R_i$ and $S_i$ the dataset and sample after processing of the first $i$ items, respectively. Since the algorithm starts with Bernoulli sampling, we have

$$\Pr[\, S_1 = \emptyset \,] = \Pr[\, S_1 = \{r_1\} \,] = 1/2$$

and

$$\begin{aligned}\Pr[\, S_2 = \emptyset \,] &= \Pr[\, S_2 = \{r_1\} \,] = \Pr[\, S_2 = \{r_2\} \,] \\ &= \Pr[\, S_2 = \{r_1, r_2\} \,] = 1/4,\end{aligned}$$

so that both $S_1$ and $S_2$ are uniform. There are two cases: (i) $|S_2| < M$ and (ii) $|S_2| = M$; these cases occur with respective probabilities $3/4$ and $1/4$. In case (i), Bernoulli sampling is continued when processing $r_3$ and we have, e.g.,

$$\begin{aligned}\Pr[\, S_3 = \emptyset, |S_2| < M \,] &= \Pr[\, S_3 = \emptyset, S_2 = \emptyset \,] \\ &= \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8},\end{aligned}$$

so that

$$\begin{aligned}&\Pr\big[\, S_3 = \emptyset \;\big|\; |S_2| < M \,\big] \\ &\quad = \Pr\big[\, S_3 = \emptyset, |S_2| < M \,\big] / \Pr[\, |S_2| < M \,] \\ &\quad = \frac{1}{8} \div \frac{3}{4} = \frac{1}{6}.\end{aligned}$$

Similar calculations show that

$$\begin{aligned}&\Pr\big[\, S_3 = \{r_1\} \;\big|\; |S_2| < M \,\big] \\ &\quad = \Pr\big[\, S_3 = \{r_2\} \;\big|\; |S_2| < M \,\big] \\ &\quad = \Pr\big[\, S_3 = \{r_3\} \;\big|\; |S_2| < M \,\big] \\ &\quad = \Pr\big[\, S_3 = \{r_1, r_3\} \;\big|\; |S_2| < M \,\big] \\ &\quad = \Pr\big[\, S_3 = \{r_2, r_3\} \;\big|\; |S_2| < M \,\big] = 1/6.\end{aligned}$$

Observe that, conditionally on $|S_2| < M$ (so that the switch-over has not yet occurred), the sample $S_3$ is not uniform, because sample $\{r_1, r_2\}$ is chosen with probability 0 whereas samples $\{r_1, r_3\}$ and $\{r_2, r_3\}$ are each chosen with probability $1/6 > 0$. For case (ii), where $|S_2| = M = 2$ (specifically, $S_2 = \{r_1, r_2\}$), reservoir sampling commences, so that

$$\begin{aligned}&\Pr\big[\, S_3 = \{r_1, r_2\} \;\big|\; |S_2| = M \,\big] \\ &\quad = \Pr\big[\, S_3 = \{r_1, r_3\} \;\big|\; |S_2| = M \,\big] \\ &\quad = \Pr\big[\, S_3 = \{r_2, r_3\} \;\big|\; |S_2| = M \,\big] = 1/3.\end{aligned}$$

We can now uncondition on $|S_2|$ to obtain

$$\begin{aligned}&\Pr[\, S_3 = \{r_1, r_2\} \,] \\ &\quad = \Pr[\, S_3 = \{r_1, r_2\} \mid |S_2| < M \,] \cdot \Pr[\, |S_2| < M \,] \\ &\qquad + \Pr[\, S_3 = \{r_1, r_2\} \mid |S_2| = M \,] \cdot \Pr[\, |S_2| = M \,] \\ &\quad = 0 \cdot \frac{3}{4} + \frac{1}{3} \cdot \frac{1}{4} = 1/12\end{aligned}$$

and, by similar calculations,

$$\Pr[\, S_3 = \{r_1, r_3\}\,] = \Pr[\, S_3 = \{r_2, r_3\}\,] = 5/24.$$

Since these two probabilities differ, $S_3$ does not constitute a uniform random sample of $R$. Note that, even when conditioned on the event $|S_3| = M$, i.e., on the event that the switch-over has occurred, the sample $S_3$ is not uniform.

The reason for the non-uniformity can be explained as follows. For $m \geq 1$, denote by $T(m)$ the first sampling step such that the sample size is equal to $m$: $T(m) = \inf\{\, i \geq 1 \colon |S_i| = m \,\}$. If we are processing a stream of insertions using Bernoulli sampling, then $S_1, S_2, \ldots$, is certainly a sequence of Bernoulli, hence uniform, samples. However, the randomly selected sample $S_{T(M)}$ is not uniform: specifically, since the sample size has just increased from $M-1$ to $M$, it follows that $\Pr\left[\, r_{T(M)} \in S_{T(M)} \,\right] = 1$. If we then apply reservoir sampling to produce the sequence $S_{T(M)+1}, S_{T(M)+2}, \ldots$, then the samples in this sequence will be non-uniform since we are applying an algorithm that assumes initial uniformity when this initial condition fails to hold.

### 2.2 Alternative: Bernoulli sampling with purging

The problem above can be avoided by avoiding the switch-over to reservoir sampling: The "Bernoulli sampling with purging" scheme—denoted $\mathrm{BSP}(q_0, M)$—is based on Bernoulli sampling only; the key idea is to incrementally reduce the sampling rate $q$ to maintain the upper bound $M$. In the context of set sampling, $\mathrm{BSP}(M)$ is equivalent to the "concise sampling" scheme of [13]. In more detail, $\mathrm{BSP}(q_0, M)$ initially runs Bernoulli sampling with sampling rate $q = q_0$. Whenever the sample size exceeds $M$, $\mathrm{BSP}(q_0, M)$ performs a *purge step*: It reduces the overall sampling rate to some value $q' < q$ (e.g., $q' = 0.8q$); each item in the sample is retained with probability $q'/q$ and rejected with probability $1 - q'/q$, independently of the other sample items. If the sample size is less than or equal to $M$ after the purging step, $\mathrm{BSP}(q_0, M)$ will recommence using $\mathrm{Bern}(q')$ sampling; otherwise, another purging step is run. $\mathrm{BSP}(q_0, M)$ appears to produce uniform samples under the insertion-only regime, although correctness has not been established formally. In any case, the sampling scheme given in Sec. 2.4 improves on $\mathrm{BSP}(q_0, M)$ by combining Bernoulli sampling with reservoir sampling, which leads to improved sample size stability and provably uniform samples. As shown in Sec. 3.4, the $\mathrm{BSP}(q_0, M)$ scheme does not yield uniform samples in the presence of deletions.

### 2.3 Alternative: Bernoulli sampling with probabilistic sample size bounds

A different, correct approach to avoiding non-uniformity in $\mathrm{HBS}(q, M)$—denoted $\mathrm{PBS}(M, \delta)$—is to settle for providing probabilistic bounds, rather than absolute bounds, on the sample size. Such bounds are not guaranteed to hold all the time but the probability of failure is small and tightly controlled. The trick is to make the sampling rate $q$ dependent on the size of the dataset, and not of the sample. In the simplest setting, the dataset size $N = |R|$ is known beforehand. The key idea is then to set the sampling rate $q$ slightly lower than $M/N$ so that the sample size exceeds $M$ with probability no more than a specified failure probability $\delta$. As shown in [4], an approximate formula for the required sampling rate is

$$q \approx \frac{N(2M + z_\delta^2) - z_\delta \sqrt{N(N z_\delta^2 + 4NM - 4M^2)}}{2N(N + z_\delta^2)}, \quad (2)$$

where $z_\delta$ is the $(1-\delta)$-quantile of the standard normal distribution. Small values of $\delta$ can be chosen without incurring too much space overhead. For example, suppose that $N = 10{,}000{,}000$, $M = 10{,}000$ and $\delta = 0.01$. The value of $q$ is then given by 0.00095 and the expected sample size is 9,500 items, which is close to $M$. Our choice of $q$ ensures that, with a probability of approximately 99%, the actual sample size will not exceed $M$.

When the dataset size is not known in advance, the value of $q$ cannot be predetermined. We can still provide probabilistic bounds as follows. We start with a high value of $q$, typically 1, and gradually reduce the sampling rate $q$ as the dataset grows [10]. Subsampling is used to reduce $q$, but in contrast to $\mathrm{BSP}(q_0, M)$, subsampling is initiated whenever the *dataset size* exceeds certain values. For example, the sampling fraction might be readjusted when the dataset size has grown by a specified amount (10,000 insertions, 1%, and so on) after the last readjustment. In general, we can choose any non-increasing function $f$ whose range is $[0, 1]$ and set $q_i = f(|R_i|)$; we execute a subsampling operation whenever $q_i$ decreases. For practical schemes, we want to choose the value of $q$ conservatively at each subsampling step. That is, the value of $N$ used in its derivation is set to the dataset size at which the next subsampling step is going to be initiated. This way, the success probability $1 - \delta$ of staying within the current sample-size bound can be guaranteed at all times.

### 2.4 Alternative: Hybrid Bernoulli sampling with randomized switch-over

An alternative approach for the insertion-only setting explicitly avoids the problem discussed in Sec. 2.1 by modifying the switch-over procedure as follows; the resulting algorithm is denoted HBSR$(q, M)$. Starting with an empty sample, perform Bern$(q)$ sampling until the sample size is $M+1$; this will occur at random time $T(M+1)$. (To simplify notation, we write $T = T(M+1)$ in the sequel.) By our prior discussion, we know that the sample $S_T$ is of the form $\{ r_T \} \cup S_{T-1}$, where $r_T$ is the item that has just been inserted into the sample. Moreover, $S_{T-1}$ is a sample of size $M$ from $R_{T-1}$, and symmetry considerations (which can be formalized) show that $S_{T-1}$ is in fact a uniform sample from $R_{T-1}$. Thus we can switch to reservoir sampling, starting with item $r_T$, and uniformity will be preserved.

Thus, we avoid the non-uniformity problem of the naive switch-over approach by further randomizing the time at which reservoir sampling starts. In the naive approach, we start reservoir sampling at $T(M)$, whereas in the modified approach, we wait for an additional, random number $T(M+1) - T(M)$ of sampling steps. The key properties of this "randomized switching" scheme are given by the following result, whose proof is in the Appendix.

**Theorem 1** *Suppose that $R_1, R_2, \ldots$ are the successive states of a database $R$ that result from a stream of insertion-only transactions, and that $S_1, S_2, \ldots$ are the successive states of a sample $S$ of $R$ that is maintained using an HBSR$(q, M)$ scheme with $q \in (0, 1)$ and $M \geq 1$. Then*

1. $\Pr[S_k = A \mid T \leq k] = \binom{k}{M}^{-1}$ *for any $k \geq 1$ and $A \subseteq R_k$ with $|A| = M$.*
2. $\Pr[S_k = A \mid T > k] = q^{|A|}(1-q)^{k-|A|} / \sum_{l>k} \pi_l$ *for any $k \geq 1$ and $A \subseteq R_k$ with $|A| < M$.*
3. *For all $k \geq 1$, the set $S_k$ is a uniform sample from $R_k$.*

*Here $T$ is the random switch-over time and $\pi_l = \Pr[T = l] = \binom{l-1}{M} q^{M+1}(1-q)^{l-1-M}$ for $l \geq M+1$.*

To illustrate the first assertion of the theorem using our previous example, assume that $T = 3$ and denote by $S_3'$ the sample right before running the reservoir step. Observe that $\Pr\big[\, S_3' = \{r_1, r_2, r_3\} \;\big|\; T = 3 \,\big] = 1$ so that $\Pr\big[\, S_2 = \{r_1, r_2\} \;\big|\; T = 3 \,\big] = 1$. We obtain $S_3$ from $S_2$ by processing $r_3$ with a standard reservoir step so that

$$\begin{aligned}
&\Pr\big[\, S_3 = \{r_1, r_2\} \;\big|\; T = 3 \,\big] \\
&\quad = \Pr\big[\, S_3 = \{r_1, r_3\} \;\big|\; T = 3 \,\big] \\
&\quad = \Pr\big[\, S_3 = \{r_2, r_3\} \;\big|\; T = 3 \,\big] = \frac{1}{3}.
\end{aligned}$$

Thus $S_3$ is a uniform sample, given that $T = 3$. By the correctness of reservoir sampling [14], it follows that, conditional on $T = 3$, each of $S_4, S_5, \ldots$ is a uniform sample of size $M$. Here $S_i$ is obtained from $S_{i-1}$ via a reservoir sampling step.

The second assertion of the theorem implies that, before the switch-over time $T$, the sample is not a true Bernoulli sample (even though we are using a "corrected" switch-over time). Indeed, the conditional probability that $S_k = A$ equals the "pure" Bernoulli probability $q^{|A|}(1-q)^{k-|A|}$ divided by the factor $\sum_{l>k} \pi_l$. The probability depends on $A$ only through $|A|$, however, and so $S_k$ is conditionally a uniform sample. The final assertion of the theorem is that uniformity actually holds unconditionally.

One key issue not addressed so far is how to choose the Bernoulli sampling rate $q$ so as to extend Bernoulli sampling as long as possible while avoiding an underfull sample. This issue, however, is analogous to the problem of how to choose $q$ when imposing probabilistic bounds on the sample size, and the discussion in Sec. 2.3 applies.

## 3 Deletions

We now discuss bounded-size set-sampling methods in the setting of general UDI transaction sequences. Reservoir sampling can be adapted to deal with deletions using a "random pairing" technique. On the other hand, techniques such as hybrid Bernoulli sampling with randomized switch-over and Bernoulli sampling with purging—both of which yield correct results in the insertion-only setting—fail in the general UDI scenario.

### 3.1 Incorrect: Adapted reservoir sampling

Tao et al. [15] propose an "adapted" reservoir sampling algorithm, denoted by ARS$(M)$, that tries to handle deletions. This algorithm maintains an array RS of length $M$, in which elements tagged as "valid" correspond to items in the sample. Whenever an item is deleted from the dataset, the item is also removed from the sample, if present, by tagging the corresponding element in RS as "invalid." When an item is inserted into the dataset, it is assigned a random integer $J$ uniformly distributed in $[1, n_I]$, where $n_I$ is the total number of insertions so far (including the newly inserted record). If $J > M$, then the record is ignored. Otherwise, the algorithm adds the item to the sample by writing it to position $J$ in RS, overwriting any prior contents at this position, and then tagging RS$[J]$ as "valid." Depending on the prior contents of RS$[J]$, such an insertion might

correspond either to simple addition of the item in the sample or to replacement by the item of a previous sample element.

Unfortunately, this algorithm does not yield uniform samples even for insertion-only transaction sequences. For example, suppose that $M = 2$ and the transaction sequence is $\gamma = (+r_1, +r_2, +r_3)$. A straightforward calculation shows that

$$\begin{aligned}
\Pr[\, S_3 = \{\, r_1 \,\}\,] &= 0 \\
\Pr[\, S_3 = \{\, r_2 \,\}\,] &= 1/6 \\
\Pr[\, S_3 = \{\, r_3 \,\}\,] &= 1/6 \\
\Pr[\, S_3 = \{\, r_1, r_2 \,\}\,] &= 1/6 \\
\Pr[\, S_3 = \{\, r_1, r_3 \,\}\,] &= 1/6 \\
\Pr[\, S_3 = \{\, r_2, r_3 \,\}\,] &= 1/3,
\end{aligned}$$

which is clearly non-uniform. In this particular counterexample, the first item $r_1$ is written into position 1 with probability 1. Item $r_2$ then either overwrites $r_1$ or is written into position 2; in either case, the sample no longer contains $r_1$ on its own. In contrast, there is a positive probability that $r_2$ appears on its own—if it overwrites $r_1$ and then $r_3$ is not accepted into the sample—and similarly for $r_3$.

A corrected variant of this algorithm is given in [10, Sec. 3.5.1F]. Experiments in [10] show, however, that this variant is dominated by the random pairing algorithm described below, which is both less expensive and yields a larger average sample size. We therefore do not consider the ARS($M$) approach further.

### 3.2 Alternative: Random pairing

The foregoing problems can be avoided by using an algorithm called random pairing [12], denoted RP($M$), briefly described here. As before, the deletion of an item is handled by removing the item from the sample, if present. As a consequence, the sample size may drop to a value less than the desired size $M$. To avoid undersized samples, RP makes use of future insertions to "compensate" for deletions. In more detail, RP keeps track of the number $d$ of deletions that have not yet been compensated. Insertions are processed as follows. If $d = 0$, so that there are no uncompensated deletions, then RP executes a standard reservoir sampling step. If $d > 0$, the decision of whether or not to include the inserted item into the sample is randomized. The inclusion probability $p$ is carefully chosen so that the uniformity of the sample is maintained; specifically, $p = \big(\min(M, |R| + d) - |S|\big)/d$. After the insertion has been processed, one deletion has been compensated and $d$ is decremented by one.

RP maintains the following invariant [12]: At any time, the sample size follows a hypergeometric distribution with parameters depending only on the values of $|R|$, $d$, and $M$:

$$\Pr[\, |S| = k \,] = \binom{|R|}{k}\binom{d}{M-k} \Big/ \binom{|R|+d}{M}. \tag{3}$$

Since the values of $|R|$ and $d$ are completely determined by the sequence of transactions, so is the sample size distribution. In particular, whenever $d = 0$, the sample size equals $M$ with probability 1.

### 3.3 Incorrect: Hybrid Bernoulli sampling with randomized switch-over and deletions

Pure Bernoulli sampling can handle UDI transactions, but the hybrid Bernoulli sampling scheme, even with randomized switch-over times and with reservoir sampling being replaced by random pairing, cannot. Specifically, suppose that we try to apply the HBSR($q, M$) algorithm to a UDI stream, and at first we see only insertions. Then, as above, the switch-over will occur at some random time $T$. Now suppose at some time $j > T$ we see a deletion transaction $-r_i$ with $i < T$. This sequence $\gamma$ of transactions is equivalent to a sequence $\gamma'$ in which element $r_i$ was never inserted into the sample at all. But for $\gamma'$, the switch-over happens at some time $T'$ that is greater than $T$.[2] In general, the bookkeeping required to roll back the sample from $j$ to $T'$ and "correct" the subsequent processing requires expensive accesses to the dataset, and appears to be impractical. (We do not give an explicit counterexample here since HBSR($q, M$) has not been proposed in prior literature.)

### 3.4 Incorrect: Bernoulli sampling with purging and deletions

In the following, we show that the BSP($q_0, M$) scheme produces non-uniform samples in the presence of deletions. In the context of set sampling, BSP($q_0, M$) essentially coincides with the concise-sampling and counting-sampling schemes of [13], which therefore also yield non-uniform samples for UDI transaction sequences. To simplify the discussion, we assume that $q' = pq$ for a fixed constant $p \in (0, 1)$; similar arguments apply when $q'/q$ can vary over the subsampling steps.

The purge operation is executed whenever the sample size increases to $M+1$, due to an accepted insertion

[2] More precisely, $T'$ is "stochastically larger" than $T$ in that $\Pr[\, T' > n \,] > \Pr[\, T > n \,]$ for all $n \geq 0$.

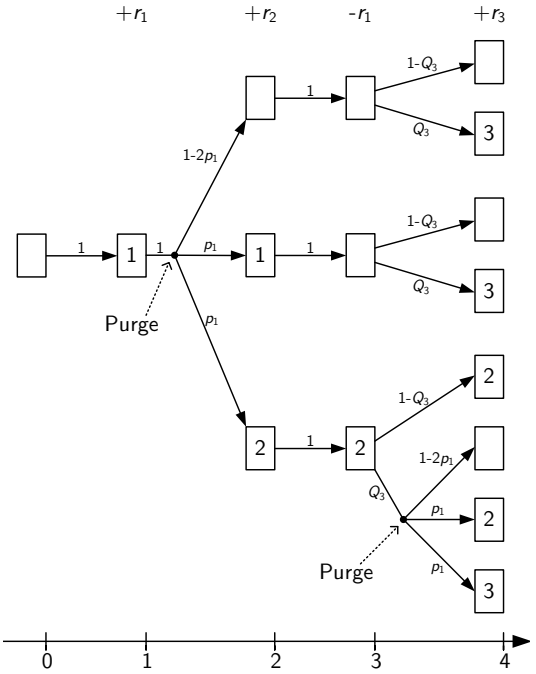


**Figure 1** Counterexample for counting sampling on sets (BSP(1, 1))

transaction. Each purge involves $L$ Bernoulli subsampling steps with sampling rate $p$, where $L$ is a geometrically distributed random variable with

$$\Pr[\,L = k\,] = p'(1-p')^{k-1}$$

for $k \geq 1$. Here, $p' = 1 - p^{M+1}$ denotes the probability that at least one of the $M$ sample items is rejected so that the purge operation terminates. Denoting by $S'$ the subsample that results from executing the purge operation on $S$, we have for any $A \subset S$

$$\begin{aligned}\Pr[\,S' = A\,] &= \Pr[\,\text{all } r \in A \text{ retained, all } r \in S \setminus A \\ &\qquad \text{purged} \mid \geq 1 \text{ item purged}\,] \\ &= \frac{p^{|A|}(1-p)^{M+1-|A|}}{p'}. \end{aligned} \tag{4}$$

After the purge operation has terminated, the sampling process proceeds with the new sampling rate $qp^L$.

We now give a simple example where BSP$(q_0, M)$ does not produce a uniform sample; an illustration is given in Fig. 1. Consider the sequence $\gamma = (+r_1, +r_2, -r_1, +r_3)$ and set $q_0 = 1$, $M = 1$. Denote by $R_i$ the dataset, by $S_i$ the sample and by $Q_i$ the (random) sampling rate after processing the $i$th transaction, starting with $R_0 = S_0 = \emptyset$ and $Q_0 = 1$. The first insertion $+r_1$ is directly included into the sample; the sampling rate remains unmodified, $Q_1 = 1$. The insertion of $r_2$ triggers a purge operation and, using (4) with $A = \{\, r_1 \,\}$, we find that $r_2$ is selected as the sample item with probability $p_1 = p(1-p)/(1-p^2)$. The same holds for $r_1$, and the sample becomes empty with probability $1 - 2p_1$. Also, the sampling rate is adjusted depending on the number $L$ of purges so that $Q_2 = p^L$. Transaction $-r_1$ simply removes $r_1$ if present in the sample; $Q_3 = Q_2$. Transaction $+r_3$ is accepted with probability $Q_3 = p^L$ and rejected otherwise. In the case of a rejection, the sample remains unmodified. Otherwise, if $r_3$ is accepted, it is included into the sample and, when additionally $S_3 = \{\, r_2 \,\}$, another purge operation is triggered. By multiplying the probabilities along the paths in Fig. 1, summing up and replacing $Q_3$ by $p^L$, we obtain

$$\begin{aligned}\Pr[\,S_4 = \{\, r_2 \,\} \mid L\,] &= p_1(1-Q_3) + p_1 Q_3 p_1 \\ &= p^L(p_1^2 - p_1) + p_1,\end{aligned}$$

and

$$\begin{aligned}\Pr[\,S_4 = \{\, r_3 \,\} \mid L\,] &= (1-2p_1)Q_3 + p_1 Q_3 + p_1 Q_3 p_1 \\ &= p^L(p_1^2 - p_1 + 1).\end{aligned}$$

We can now uncondition on $L$ and simplify:

$$\begin{aligned}\Pr[\,S_4 = \{\, r_3 \,\}\,] &= \sum_{k=1}^{\infty} \Pr[\,L = k\,] \Pr[\,S_4 = \{\, r_3 \,\} \mid L = k\,] \\ &= \sum_{k=1}^{\infty} (1-p^2)(p^2)^{k-1} p^k (p_1^2 - p_1 + 1) \\ &= \frac{1-p^2}{p^2}(p_1^2 - p_1 + 1) \sum_{k=1}^{\infty} (p^3)^k \\ &= \frac{p}{p+1}.\end{aligned}$$

Similarly,

$$\begin{aligned}\Pr[\,S_4 = \{\, r_2 \,\}\,] &= \sum_{k=1}^{\infty} \Pr[\,L = k\,] \left(p^k(p_1^2 - p_1) + p_1\right) \\ &= \frac{p}{p+1} \frac{p^2+1}{p^2+p+1}.\end{aligned}$$

It follows that

$$\Pr[\,S_4 = \{\, r_2 \,\}\,] < \Pr[\,S_4 = \{\, r_3 \,\}\,]$$

for $p > 0$, so that BSP$(q_0, M)$ biases the sample toward recent items. For example, a common choice is $p = 0.8$; the two probabilities are then given by $\approx 0.30$ and $\approx 0.44$, respectively. The purge operation thus introduces some subtle dependencies among the sample items, and these dependencies lead to non-uniform samples when the transaction sequence contains deletions.

As a final note, Tao et al. [15] proposed an extension of BSP$(q_0, M)$—called adapted counting sampling, ABSP$(M)$—that attempts to avoid the cost of

executing purging steps by continually adjusting the sampling rate. This can be shown to produce non-uniform samples even in the insertion-only setting. A simple counterexample is provided by the sequence $\gamma = (+r_1, +r_2, +r_3)$ with $M = 1$, which can be shown to yield

$$\Pr[\,S_3 = \{\,r_1\,\}\,] = 3/8$$
$$\Pr[\,S_3 = \{\,r_2\,\}\,] = 3/8$$
$$\Pr[\,S_3 = \{\,r_3\,\}\,] = 1/4,$$

a clearly non-uniform result.

### 3.5 Alternative: Bernoulli Sampling with probabilistic sample size bounds

The PBS($M$) scheme with probabilistic bounds given in Sec. 2.3 extends to the case of deletions. The idea is to choose a non-increasing function $f$ whose range is $[0, 1]$, set $q_i = f(\max_{j \le i} |R_i|)$, and execute a subsampling operation whenever $q_i$ decreases. This extension is the only Bernoulli-based scheme that supports arbitrary UDI transactions. As before, the sequence $\{q_i\}$ is a deterministic decreasing sequence, given the sequence of transactions, and at any point in time, the current sample is true Bern($q_i$) sample. Because Bernoulli sampling supports deletion transactions, the sample remains uniform under such deletions. Note however, that we cannot increase $q$ when the dataset shrinks due to deletions.

## 4 Multiset Sampling Methods

In the foregoing discussion, we have assumed that, after every transaction, the dataset $R$ and sample $S$ are both true sets, i.e., containing no duplicate items. We now focus on the case in which $R$ and $S$ are multisets, so that they can contain duplicates, and items having the same value are indistinguishable. In this setting, we can save space by storing the sample in a compressed format. The *compressed sample* $H$ of $S$ contains a pair $(r, n_r)$ for every distinct $r \in S$, where $n_r = |S(r)|$ is the multiplicity of $r$ in $S$. Thus we store (item, frequency)-pairs in $H$, instead of repeatedly storing duplicate items.[3] In what follows, we refer to $|S|$ as the sample size (including duplicates) and $|H| = |D(S)|$ as the *sample footprint* (excluding duplicates); note that $|H| \le |S|$.

Importantly, for an insertion-only transaction sequence, we can apply any of the uniform set-sampling algorithms that work in this scenario, regardless of the format in which we store the sample—whether we insert an accepted sample value into a sample array or increment a counter for the value is immaterial. This approach does not work in the UDI setting, however. Suppose, for example, that a transaction of the form $-4$ arrives. We clearly need to decrement by 1 the number of 4's in the dataset. It is not clear, however, whether or not to decrease the number of 4's in the sample. Multiset sampling methods deal with this question in a manner that guarantees uniformity. Some multiset sampling methods attempt to provide strict bounds directly on the sample footprint. As shown in the following sections, such strict bounds always lead to non-uniformity, and only probabilistic bounds are available. To avoid confusion between bounded-size and bounded-footprint methods, we consistently denote sample-size bounds by $M$ (including duplicates) and sample-footprint bounds by $F$ (excluding duplicates).

[3] Even more space can be saved by representing a singleton value $(r, 1)$ simply as $(r)$; see [13]. For simplicity, we ignore this refinement here.

### 4.1 Incorrect: Bernoulli sampling with purging and bounded footprint

Denote by BSP($q_0, F$) the variant of Bernoulli sampling with purging in which subsampling is triggered when a specified sample footprint is exceeded. Brown and Haas [4] observed that BSP($q_0, F$), and indeed any algorithm that attempts to strictly bound the sample footprint, must yield non-uniform samples, even for an insertion-only sequence. To see this, suppose that the dataset is empty initially and consider a transaction sequence $\gamma = (+r_1, +r_1, +r_2)$. Suppose that $F = 1$, so that only one (item, frequency)-pair can be stored in the sample, and consider the samples $S_1 = \{\,r_1, r_1\,\}$ and $S_2 = \{\,r_1, r_2\,\}$. The corresponding compressed samples are given by $H_1 = \{\,(r_1, 2)\,\}$ and $H_2 = \{\,(r_1, 1), (r_2, 1)\,\}$. If BSP were uniform, we would have $\Pr[\,S_2\,] = 2\Pr[\,S_1\,] > 0$ or $\Pr[\,S_2\,] = 2\Pr[\,S_1\,] = 0$. By inspection, we can see that $H_1$ has a positive probability of being produced, whereas $H_2$ is never produced since it exceeds the footprint $F = 1$. This argument applies to any multiset sampling scheme that strictly bounds the footprint.

The concise-sampling and counting-sampling schemes in [13] enforce strict bounds on the sample footprint, and hence yield non-uniform samples in the multiset context, even for insertion-only sequences. In the set-sampling scenario, the footprint coincides with the sample size, so these algorithms will yield uniform samples for an insertion-only transaction sequence. (Recall from Sec. 3.4 that these algorithms fail to produce uniform samples in the presence of deletions.)

### 4.2 Alternative: Augmented Bernoulli sampling with probabilistic sample size bounds

In the UDI setting, we can obtain a uniform sampling algorithm with a probabilistically bounded sample size and compressed sample format based on the augmented Bernoulli sampling ABS($q$) scheme given in [11]. The ABS($q$) algorithm borrows an idea from [13] and maintains for each item $r$ in the sample both $X_i(r) = |S_i(r)|$, the frequency of $r$ in the sample, and a "tracking counter" $Y_i(r)$. Whenever $X_i(r)$ is positive, the counter $Y_i(r)$ records the number of net insertions of $r$ into the dataset that have occurred since the insertion of the first of the current $X_i(r)$ sample items; the dataset insertion corresponding to the first of these $X_i(r)$ sample inclusions is counted as part of $Y_i(r)$.

The general layout of the sample $S_i$ is as follows: for each distinct item $r \in R$ that occurs in the sample at least once, $S_i$ contains the triple $(r, X_i(r), Y_i(r))$; the sample is therefore "augmented" with tracking counters. To save space, the entry for $r$ is stored as $(r, X_i(r), Y_i(r))$ if $Y_i(r) > 1$ and simply as $(r)$ if $X_i(r) = Y_i(r) = 1$.

Recall that, in Bernoulli sampling, each item is sampled independently from all the other items. Without loss of generality, therefore, fix an item $r$ and focus on the maintenance of $X_i(r)$ and $Y_i(r)$ as a transaction sequence $\gamma$ is processed—assume that $\gamma$ consists solely of insertions and deletions of item $r$. We represent the state of $S_i$ as $(X_i, Y_i)$, that is, we suppress the dependence on $r$ in our notation. We have $X_i = Y_i = 0$ whenever $r \notin S_i$. As before, we assume that both the dataset and the sample are initially empty so that $X_0 = Y_0 = 0$.

The algorithm works as follows: For an insertion transaction $\gamma_{i+1} = +r$, set

$$(X_{i+1}, Y_{i+1}) \leftarrow \begin{cases} (X_i + 1, Y_i + 1) & \text{if } \Phi_{i+1} = 1 \\ (X_i, Y_i + 1) & \text{if } \Phi_{i+1} = 0, X_i > 0 \\ (0, 0) & \text{otherwise,} \end{cases}$$

where $\Phi_{i+1}$ is a 0/1 random variable such that $\Pr[\,\Phi_{i+1} = 1\,] = q$. For a deletion $\gamma_{i+1} = -r$, set

$$(X_{i+1}, Y_{i+1}) \leftarrow \begin{cases} (0, 0) & \text{if } X_i = 0 \\ (0, 0) & \text{if } X_i = Y_i = 1 \\ (X_i - 1, Y_i - 1) & \text{if } X_i \geq 1, Y_i > 1, \Psi_{i+1} = 1 \\ (X_i, Y_i - 1) & \text{otherwise,} \end{cases}$$

where $\Psi_{i+1}$ is a 0/1 random variable such that

$$\Pr[\,\Psi_{i+1} = 1\,] = \frac{X_i - 1}{Y_i - 1}.$$

Note that $\Pr[\,\Psi_{i+1} = 1\,] = 0$ whenever $X_i = 1$; we have $X_{i+1} \geq 1$ whenever $Y_i > 1$. As before, item $r$ is removed from the sample if $X_i > 0, X_{i+1} = 0$ and added to the sample if $X_i = 0, X_{i+1} > 0$.

Although this algorithm does not provide any bounds on the sample size, we can proceed as in Sec. 2.3, initially setting $q = 1$ and then periodically reducing $q$ and purging the sample when the dataset size (including duplicates) exceeds specified bounds. We refer to this variant of ABS($q$) as PABS($F, \delta$). Here parameter $F$ bounds the footprint of the sample probabilistically, with exceedance probability $\delta$. The sample size $\sum_{r \in S} X_i(r)$ is also bounded probabilistically by $F$, but the number of net insertions $\sum_{r \in S} Y_i(r)$ is not.

## 5 Distinct-Item Sampling

Recall that the goal of distinct-item sampling is to obtain a uniform sample of $D(R)$, the set of distinct items in the dataset. Most (but not all) of the existing sampling techniques in this setting make use of a random hash function as source of randomness.

Denote by $\mathscr{H} = \{\, h_1, \ldots, h_{|\mathscr{H}|} \,\}$ a class of hash functions with domain $\mathscr{D}$ and range $\mathscr{B}$. Following [16], we say that $\mathscr{H}$ is *$k$-wise independent* if for any $k$ distinct elements $a_1, \ldots, a_k \in \mathscr{D}$ and $k$ not necessarily distinct elements $b_1, \ldots, b_k \in \mathscr{B}$, there are exactly $|\mathscr{H}|/|\mathscr{B}|^k$ functions $h \in \mathscr{H}$ such that $h(a_i) = b_i$ for $1 \leq i \leq k$. For example, let $p$ be an arbitrary prime number and denote by $Z_p$ the finite field over $\{\, 0, 1, \ldots, p-1 \,\}$ with integer addition and multiplication modulo $p$. Then

$$\mathscr{H}_p = \{\, h(x) = (ax + b) \bmod p : a, b \in \{\, 0, \ldots, p-1 \,\} \,\} \tag{5}$$

is pairwise independent (i.e., $k = 2$) with $\mathscr{D} = \mathscr{B} = \{\, 0, 1, \ldots, p-1 \,\}$. Finally, say that $\mathscr{H}$ is *fully independent* if it is $k$-wise independent for all values of $k$; only the set of all functions from $\mathscr{D}$ to $\mathscr{B}$ is fully independent.

In what follows, we assume that $D(R) \subseteq \mathscr{D}$ at all times, that $\mathscr{B} = \{\, 0, \ldots, |\mathscr{B}| - 1 \,\}$, and that $h$ is picked uniformly and at random from $\mathscr{H}$. When using hashing for distinct-item sampling, we will need to store hash function $h$ in addition to the sample. If $\mathscr{H}$ is pairwise independent, it suffices to store the integers $a$ and $b$ of Eq. (5). If $\mathscr{H}$ is fully independent, we essentially have to store the hash value of every item $r \in D(R)$, which is prohibitive. In practice, however, cryptographic hash functions (e.g., based on AES) often "look" fully independent and may be used to approximate fully inde-

pendent hashing [10] for practical purposes, although no theoretical guarantees are provided.

Most of the sampling techniques discussed below have rather strong independence requirements on the class of hash functions. As outlined above, the higher the requirements on independence, the more cost is incurred for storing and, in many cases, applying the hash function. It is thus of practical interest to make use of $k$-wise independent hash functions for some small value of $k$. The only known sampling scheme that uses pairwise-independent hashing is the dynamic inverse sampling scheme of [6]; we show below that this scheme is not uniform. We also briefly outline a number of other distinct-item sampling techniques proposed in the literature; in particular, the PABSD$(F, \delta)$ scheme is the only sampling scheme that does not make use of hashing.

### 5.1 Incorrect: Dynamic Inverse Sampling with pairwise-independent hashing

Cormode et al. [6] proposed a strictly bounded-size scheme for distinct-item sampling. The scheme—called dynamic inverse sampling and denoted DIS$_2(F)$—can handle arbitrary insertions and deletions. It makes use of $F$ data structures and $F$ pairwise-independent hash functions, one per data structure. Each data structure maintains—with some success probability $p > 0$—a *single* item chosen uniformly and at random from the set $D(R)$ of distinct items in $R$. The sample size is thus random in $[0, F]$ and sampling is with replacement.

Each data structure consists of $O(\log|\mathscr{D}|)$ buckets. Each inserted or deleted item affects exactly one of the buckets in a data structure; the hash function associated with the data structure ensures that each item maps to the same bucket whenever it occurs in the transaction sequence. In more detail, an item is hashed to bucket $i$ with probability $(1-\alpha)\alpha^{i-1}$, where $0 < \alpha < 1$ is a parameter of the algorithm.[4] Each bucket consists of the *sum* of the items (treated as integers) inserted into it, a *counter* of the number of inserted items, and a data structure for *collision detection*. Adding an item to (resp., deleting an item from) a bucket simply involves incrementing (resp., decrementing) the counter by 1, incrementing (resp., decrementing) the sum by the item value and updating the collision detection structure accordingly. If there is a bucket in the data structure that contains exactly one distinct item, then the data structure "succeeds," and this item is returned as a random sample of size 1; otherwise, the data structure fails. Note that the lower-numbered buckets are more likely to succeed when the dataset size is small; the higher-numbered buckets handle large datasets. The point is that the stored items do not need to be maintained individually; in the important case where a bucket contains only a single distinct item (as indicated by the collision-detection data structure), the value of the sum is equal to the value of the counter times the value of the item; the item can thus be extracted. [6] have shown that, for an appropriate choice of $\alpha$, the success probability $p$ is at least 14.2%.

[4] The bucket number of item $r$ is given by $\lceil \log_{1/\alpha}(|\mathscr{B}|/[h(r)+1]) \rceil$.

**Table 2** Frequencies of each possible sample for DIS$_2(1)$ after $N$ database insertions

| | $N=2$ | $N=3$ |
|---|---|---|
| $S=\emptyset$ | 6,776,653 | 3,388,337 |
| $S=\{0\}$ | 30,157,914 | 23,029,192 |
| $S=\{1\}$ | 30,157,914 | 17,645,760 |
| $S=\{2\}$ | | 23,029,192 |
| Uniform? | ✓ | - |

Because a DIS$_2(F)$ data structure is successful if there exists a bucket comprising a single distinct item, and because this event depends on the hash values of all items in $R$, one might suspect that the use of pairwise-independent hash functions leads to non-uniform samples. A simple setup can be used to show that DIS$_2(F)$ is indeed non-uniform: Pick a pairwise-independent class of hash functions, run DIS$_2(F)$ on some dataset $R$ for all $h \in \mathscr{H}$, and test whether every sample of the same size is produced by exactly the same number of hash functions. Table 2 shows results for DIS$_2(1)$, $\mathscr{H} = \mathscr{H}_{8191}$, $R_2 = \{0,1\}$, and $R_3 = \{0,1,2\}$.[5] As can be seen, DIS$_2(F)$ is uniform for $R_2$, but it is non-uniform for $R_3$.

A simple fix to DIS$_2(F)$ would be to make use of fully independent hash functions, and we denote by DIS$_\infty(F)$ the resulting sampling scheme. It may be possible, however, to make use of a weaker class of hash functions. Specifically, it seems plausible that some variant of min-wise independent permutations [3] is sufficient to ensure the uniformity of DIS, in light of the discussion on min-wise hashing in Sec. 5.3 below and the fact that a succeeding DIS data structure outputs the item with the largest hash value.

### 5.2 Alternative: Distinct-item Bernoulli sampling with probabilistic bounds

Set $h'(x) = h(x)/|\mathscr{B}|$ for $h \in \mathscr{H}$, and consider the following hash-based variant of Bernoulli sampling with

[5] We used $\alpha = \sqrt{2/3}$ and the greedy version of DIS$_2(1)$. Other values of $\alpha$ as well as the basic version led to similar results.

sampling rate $q$: When processing an insertion $+r$, accept the item into the sample if $h'(r) < q$; otherwise ignore the item. When processing deletion $-r$, remove one occurrence of $r$ from the sample, if present. As before, we store the sample in compressed form. It is easy to see that if $R$ is produced by a sequence of insertions and transactions, and $S$ is obtained by running the sampling algorithm above on the same sequence, we have

$$|S(r)| = \begin{cases} |R(r)| & \text{if } h'(r) < q \\ 0 & \text{otherwise.} \end{cases}$$

If $\mathscr{H}$ is fully independent and $q = k/|\mathscr{B}|$ for some integer $0 \le k \le |\mathscr{B}|$, then

$$\Pr[\, D(S) = A \,] = q^{|A|}(1-q)^{|D(R)|-|A|};$$

thus $D(S)$ has the same distribution as a Bern$(q)$ sample of $D(R)$ and uniformity follows.

To obtain a bounded-footprint sampling scheme, we can now employ probabilistic bounds as before, i.e., we start with a high value of $q$ and gradually reduce the sampling rate as the dataset grows. We denote the resulting scheme by DPBS$(F, \delta)$. To reduce $q$ to $q' < q$, we retain items $r \in S$ with $h'(r) < q'$ and remove items with $h'(r) \ge q'$. Note that DPBS$(F, \delta)$ does not guarantee a bounded sample size; for example, if $R = \{\, r_0 \,\}^N$ and $h'(r_0) = 0$, then $|S| = N$ for all $q > 0$. DPBS$(F, \delta)$ does, however, provide bounds on the sample footprint $|D(S)|$. In particular, if we choose $q$ as in Eq. (2) (using $M = F$), then $|D(S)| \le F$ with high probability. This bound is somewhat unsatisfactory though: Duplicates do not affect the sample footprint, but $q$ is selected based on $|R|$ (i.e., including duplicates) and can thus be unnecessarily small. An alternative approach is to select $q$ based on $|D(R)|$—i.e., taking $N = |D(R)|$ in Eq. (2)—or an estimate of $|D(R)|$; if the estimate is not based on the sample, uniformity is retained.

### 5.3 Alternative: Min-wise hashing

We now briefly discuss some alternative schemes that have been proposed in the literature. In min-wise hashing [2], denoted MIN$(M)$, the sample $S$ consists of the distinct items in $R$ having the $M$ smallest hash values. The sample $S$ can be maintained incrementally in the insertion-only setting, as follows. Insert the first $M$ distinct items directly into the sample. Then, whenever an item $r^+$ is inserted into $R$ and $r^+ \notin S$, check whether $h(r^+) < h(r^-)$, where $r^- = \operatorname{argmax}_{r \in S} h(r)$. If so, add $r^+$ to the sample and remove $r^-$. The MIN$(M)$ scheme produces uniform samples if $\mathscr{H}$ is a family of $M$-minwise independent permutations [3].

Deletions can be handled by augmenting the sample with frequency counters as in DPBS$(F, \delta)$; the resulting scheme is due to Beyer et al. [1] and denoted AKMV$(F)$ (for "augmented $k$-minimum values"). The frequency counters record the exact frequency in $R$ of each sampled element, and AKMV$(F)$ retains sample elements whose frequency has dropped to 0 to ensure uniformity; all frequency-0 items are ignored for estimation purposes.

### 5.4 Alternative: Distinct-item subsampling for PABS$(F, \delta)$

PABS$(F, \delta)$ samples can also be used to obtain a distinct-item sample of $R$ under UDI transactions [11]; denote by PABSD$(F, \delta)$ the resulting scheme. In particular, let $S$ be a PABS$(F, \delta)$ sample of $R$ and denote by $q$ its current sampling rate. To obtain distinct-item sample $S'$, include $r \in D(S)$ into $S'$ with probability

$$p(r) = \begin{cases} 1 & \text{if } Y(r) = 1 \\ q & \text{if } Y(r) > 1. \end{cases}$$

Then $S'$ is a Bern$(q)$ sample of $D(R)$. PABSD$(F, \delta)$ is the only distinct-item scheme that does not make use of hashing. Note, however, that PABS$(F, \delta)$ generally has a larger footprint that DPBS$(F, \delta)$ (though both footprints are probabilistically bounded by $F$). To see this, fix some sampling rate $q$ and item $r \in R$. We have $r \in D(S)$ with probability $1 - (1-q)^{|R(r)|}$ for PABS$(F, \delta)$, and $r \in D(S)$ with probability $q$ for DPBS$(F, \delta)$. Since both samplers produce a Bern$(q)$ sample of $D(R)$, and since $1 - (1-q)^{|R(r)|} \ge q$, the expected sample footprint of PABS is larger than or equal to the one of DPBS.

## 6 Merging

One factor in choosing between sampling methods is whether samples can easily be merged, and so we briefly summarize some results on this topic. Merging is of particular interest when a dataset $R$ is partitioned across several nodes; see [4] for an example. In this case, it may be desirable to independently maintain a local sample of each partition and compute a global sample of the complete dataset (or, in general, of any desired union of the partitions) by merging these local samples. This approach is often superior, in terms of parallelism and communication cost, to first reconstructing $R$ and sampling afterward.

The easiest case is when $S_1$ and $S_2$ are Bernoulli samples of partitions $R_1$ and $R_2$ with respective sampling rates $q_1$ and $q_2$ [4]. Simply purge one of the samples so that both samples are Bernoulli with common rate $q = \min(q_1, q_2)$, and then set $S = S_1 \cup S_2$. This procedure exploits the fact that the items in the dataset are accepted into or rejected from the sample in a mutually independent fashion. The ease of parallelization of Bernoulli sampling was one of the motivations for developing hybrid sampling schemes such as $\mathrm{BSP}(q_0, M)$ and $\mathrm{HBSR}(q, M)$. Since hybrid schemes do not maintain true Bernoulli samples, however, the above merging technique cannot be used, and the parallelization advantages of Bernoulli sampling thus vanish. A notable exception is $\mathrm{PBS}(M, \delta)$, which provides probabilistic sample size bounds; here the sample constitutes a true Bernoulli sample at any time.

Brown and Haas [4] provide an algorithm, called MERGE, that is designed to merge uniform samples $S_1$ and $S_2$ into a sample $S$ in the insertion-only environment; the algorithm makes no assumptions about the method used to create the uniform samples $S_1$ and $S_2$. The MERGE algorithm as described by [4] accesses $S_1$ and $S_2$ to create a uniform sample $S$ of size $m = \min(|S_1|, |S_2|)$. The basic idea is to select $X_1$ random items from $S_1$ and $X_2 = m - X_1$ items from $S_2$ to include in $S$, with $X_1$ being hypergeometrically distributed:

$$\Pr[\, X_1 = k \,] = \binom{|R_1|}{k} \binom{|R_2|}{m-k} \Big/ \binom{|R_1| + |R_2|}{m}. \qquad (6)$$

Indeed, the right side of (6) is the probability that exactly $k$ out of $m$ random items from $R_1 \cup R_2$ belong to $R_1$. The resulting sample is therefore statistically equivalent to a size-$m$ uniform sample from $R_1 \cup R_2$ so that MERGE is indeed correct. MERGE can also be used in the deletion setting but has the disadvantage that the size of the merged sample is limited by the size of the smallest constituent sample, and hence is sensitive to skew in the local sample sizes caused by uncompensated deletions [12].

To address this problem when $S_1$ and $S_2$ are both created using the RP algorithm, Gemulla, et al. [12] provide an extension of MERGE, called RPMERGE, that yields larger merged samples and is resistant to skew; moreover, the sample is accompanied by sufficient information so that incremental maintenance can be continued.

The merging problem is more complicated for multiset sampling. In an insertion-only setting, we can use set sampling methods (see Sec. 4) and the foregoing discussion applies. In the presence of deletions, however, the $\mathrm{PABS}(F, \delta)$ algorithm is the only available sample-maintenance method. It would be desirable to be able to merge two PABS samples into a combined PABS sample, so that incremental maintenance of this latter sample can be continued. Unfortunately, as shown in [11], if multiset partitions $R_1$ and $R_2$ share any common values, then such merging is impossible without accessing the underlying dataset. The best that one can do is compute a one-time Bernoulli sample $S$ by discarding the tracking counters and setting $|S(r)| = |S_1(r)| + |S_2(r)|$ for each $r \in D(S_1 \cup S_2)$; the combined sample $S$ cannot be maintained further by a bounded uniform sampling technique (strict or probabilistic).

Similar reasoning applies to $\mathrm{PABSD}(F, \delta)$ for distinct-item sampling: We cannot merge samples if partitions share any common values. In contrast, all other distinct-item schemes support both merging and subsequent incremental maintenance when the same hash function is used to construct $S_1$ and $S_2$. For MIN and AKMV, Beyer et al. [1] propose merging schemes that can obtain $\mathrm{MIN}(M)$ and $\mathrm{AKMV}(F)$ samples of $R_1 \cup R_2$, where $M = \min(M_1, M_2)$ and $F = \min(F_1, F_2)$; here $M_i$ and $F_i$ denote the parameters of $S_i$ for $i \in \{1, 2\}$. For $\mathrm{DPBS}(F, \delta)$, we proceed similarly to Bernoulli sampling, first purging to $q = \min(q_1, q_2)$ and then taking the multiset union. Finally, for $\mathrm{DIS}_\infty(F)$ and $F = F_1 = F_2$, we sum up the counters and collision detection data structures of the corresponding buckets in $S_1$ and $S_2$.

## 7 Resizing

In this section we focus on maintaining a bounded uniform sample $S$ in the presence of a "growing" dataset $R$ in which insertions occur more frequently than deletions over the long run. A key challenge in this setting is that the relative sample size $M = |S|$ may eventually become too small relative to $|R|$, leading to an unacceptable loss of precision when using the sample. Resizing schemes aim to increase the sample-size bound from $M$ to a specified value $M' > M$ in a controlled manner, while preserving uniformity. Such resizing operations can be performed periodically as $R$ continues to grow.

### 7.1 Incorrect: Bernoulli Resizing

As mentioned previously, the Bernoulli resizing (BR) algorithm in [12] first converts the initial reservoir sample to a Bernoulli sample, then performs Bernoulli sampling (with deletions) until the sample size has reached the new upper bound, and finally switches back to reservoir

sampling. For example, suppose that the initial sample-size bound is $M = 2$ and that a sequence of insertions produces both the dataset $R = \{1, 2, \ldots, 10\}$ and a reservoir sample $S = \{2, 3\}$. To obtain a new sample of size $M' = 4$, the BR algorithm converts $S$ to a Bernoulli sample by first generating a sample size $U$ according to a Bernoulli$(N, q)$ distribution, where $q$ is a parameter of the algorithm. Supposing for our example that $U = 3$, the algorithm then creates the initial sample by augmenting $S$ with a randomly-selected element from the set $\{1, 4, 5, \ldots, 10\}$, say 7, so that $S = \{2, 3, 7\}$. Bernoulli sampling is now performed until $|S| = 4$, at which point reservoir sampling recommences. When $U \leq M$, the initial Bernoulli sample is obtained by uniformly subsampling $S$; when $U > M'$, then $S$ is augmented with a uniform sample of size $M' - M$ from $R \setminus S$. As can be seen, the BR algorithm may incur an expensive access to the underlying dataset $R$; it is shown in [12] that such accesses are unavoidable in any resizing scheme. The parameter $q$ is chosen to optimally balance the time for accessing $R$ and the time to subsequently grow the sample size up to $M'$; the goal is to incur substantially less time than would be required to recompute the sample from scratch.

By the arguments of Sec. 2.1, the scheme does not produce uniform samples. Indeed, attempting to enforce uniformity can be viewed as even more challenging in the current setting than in that of previous sections. First, since deletions are allowed, the modified switch-over scheme HBSR$(q, M)$ of Sec. 2.4 cannot help. Second, even for an insertion-only transaction sequence, there is an additional problem unique to Bernoulli resizing. Specifically, the "conversion" to a "Bernoulli" sample is not quite correct: while executing Bernoulli sampling steps during the resizing phase, we know that the sample size must be less than $M'$. Hence the sample cannot be a true Bernoulli sample, since such a sample has a size exceeding $M'$ with non-zero probability. Since the sample is not Bernoulli, application of Bernoulli "coin flips" to this sample yields erroneous results.

### 7.2 Alternative: Random Pairing Resizing

Our workaround for resizing is to not rely on Bernoulli sampling at all. The key idea of the corrected resizing algorithm is to first convert the initial sample to a random pairing sample for some value of $d$, and then run the random pairing algorithm until $d$ becomes 0. At that time, resizing is completed. Note that, in contrast to Bernoulli resizing, the number of steps in the second phase—i.e., after the conversion of the sample—is completely determined by the transaction sequence and the initial value of $d$. This guarantees the correctness of the algorithm. We also provide guidance into the choice of parameter $d$; here we use the same cost model as in [12], but found that parameter choice is simplified somewhat in RPR.

Suppose that the initial sample size is $|S| = M$ and the target sample size is $M' > M$, where $M, M' < |R|$. We say that a sample $S$ is an RP$(M', d)$ sample of $R$ if it is produced by running RP with sample-size parameter $M'$ on a sequence that produces $R$ and contains $d$ uncompensated deletions. Such a sequence exists for any value of $d$; for example, the sequence may consist of $|R|$ insertions, one for each item in $R$, followed by the insertion of $d$ "transient" items, which are subsequently deleted. As mentioned above, the key idea of random pairing resizing (RPR) is to convert sample $S$ to an RP$(M', d)$ sample, where $d$ is treated as a parameter of the conversion process. The conversion may require access to the base data, but—as we discuss in the following section—the probability and number of such accesses depend on $d$. After conversion, subsequent transactions are processed using RP so that, after a sufficiently large number of insertions, the sample size reaches its target value $M'$. When base data accesses are expensive and insertions occur frequently, this approach can be much faster than recomputing the sample from scratch.

We now describe the algorithm in more detail. In phase 1 (conversion), RPR generates a realization of a random variable $U$, which is distributed as in (3) and represents the initial RP$(M', d)$ sample size. From standard properties of the hypergeometric distribution, we obtain that $\mathrm{E}[U] = M'|R|/(|R| + d)$. For $d = 0$, we have $U = M'$ with probability 1; for larger values of $d$, the (expected) value of $U$ decreases. The algorithm now uses as many items from $S$ as possible to make up the RP$(M', d)$ sample. Base data is only accessed when $U > M$, in which case we add $U - M$ random items from $R \setminus S$ to $S$ to form the initial sample. In phase 2 (growth), the algorithm increases the sample to the desired size by running the RP algorithm on subsequent transactions. Since the dataset is growing, the sample size will eventually reach its target value $M'$. The phase ends as soon as the number of uncompensated deletions reaches 0, in which case the sample size is guaranteed to equal $M'$.

We assume that the dataset is "locked" during phase 1, so that the process of incoming transactions is temporarily suspended. For ease of exposition, we assume that the sample of $R \setminus S$ is obtained using "draw-sequential" sampling techniques. These techniques obtain a single random item from $R \setminus S$ by first extracting a random item from the dataset $R$ and accepting

the item if it is not already in the sample (which is the usual case); otherwise, the item is rejected and the process starts over. As discussed in [12], more sophisticated and efficient data-access methods may be available, depending upon the specific system architecture and data layout. Our goal, given cost models for a specified base-data access mechanism and the sequence of transactions, is to optimally balance the amount of time required to access the base data in phase 1, and the amount of time required to finish growing the sample (using new insertions) in phase 2.

The value of the parameter $d$ determines the relative time required for phases 1 and 2. Intuitively, when base data accesses are expensive but new insertions occur frequently, we might want to choose a large value of $d$ so as to resample as few items as possible and shift most of the work to phase 2. In contrast, when base data accesses are fast with respect to the arrival rate of new insertions, a small value of $d$ might be preferable to minimize the complete resizing time.

The next section addresses the key problem of choosing the parameter $d$ in the corrected resizing algorithm; our discussion parallels that in [12]. For a given choice of $d$, the resizing cost—i.e., the time required for resizing—is random. Indeed, the time required for phase 1 depends on the value of the hypergeometric random variable $U$, and the time required for phase 2 depends on both $d$ and the transaction sequence. Our goal is therefore to develop a probabilistic model of the resizing process, and choose $d$ to minimize the expected resizing cost.

We develop perhaps the simplest possible model, based on the dataset-access paradigm described previously and a very simple model of the transaction stream. A discussion of more complex cost models and their implications on the resizing cost can be found in [12]. Given a cost model, numerical methods can be used to determine $d^*$, the optimal value of $d$. Since numerical optimization can be too expensive at run time, we also consider an approximate model of the cost function that can be minimized analytically. Our experiments (see Sec. 7.4) indicate that both the approximate model of expected cost and the resulting choice of $d$ closely agree with the results obtained via numerical methods, thereby justifying the use of the quick approximate analytical method.

### 7.3 Modeling the resizing process

We first consider the cost of phase 1. During this phase, the algorithm obtains $N(U)$ items from $R \setminus S$, where

$$N(u) = \max(u - M, 0)$$

for $0 \le u \le M'$. As discussed above, we assume that these items are obtained using repeated simple random sampling from $R$ with replacement, with an acceptance-rejection step to ensure that each newly sampled item is not an element of $S$ and is distinct from all of the items sampled so far. Using a result from [12], we find that a good approximation of the expected number of base data accesses when $U = u$ is given by

$$g(u) = |R| \ln\left(\frac{|R| - M}{|R| - M - N(u)}\right). \tag{7}$$

Supposing that each base-data access takes $t_a$ time units, the expected phase 1 cost is

$$T_1(d) = t_{\mathrm{a}}\, \mathrm{E}_d\,[\, g(U)\,],$$

where we use the subscript $d$ to emphasize the fact that the expected value depends on the value of $d$.

We now consider the cost of phase 2. In this phase, the resizing algorithm executes $L$ steps of the random pairing algorithm, where $L$ depends on the transaction sequence. To make further progress, we need a model of the insertion and deletion process. The simplest model, which we will use here, is to assume that, during phase 2, a sampling step occurs every $t_{\mathrm{b}}$ time units. The quantity $t_{\mathrm{b}}$ primarily reflects the time between successive transactions. With probability $p$, the transaction is an insertion, and with probability $q = (1-p)$ the transaction is a deletion. We assume that $p > 1/2$, since the dataset is growing. The parameters $t_{\mathrm{b}}$ and $p$ can easily be estimated from observations of the arrival process.

The distribution of the number $L$ of transactions in phase 2 can be obtained by an analogy to a ruin problem, see for example [8]. In the classical ruin problem, a gambler wins or loses a dollar with probability $p$ and $q = 1 - p$, respectively. The gambler is given initial capital $z$ and the game ends when the gambler's capital reduces to zero (=ruin) or reaches value $a$ (=win). We are interested in the expected number of steps until the gambler either wins or is ruined. In our setting, we have $z = |R|$ and a=$|R| + d$. The expected value of the duration of the ruin problem is given by [8, p. 348]

$$\mathrm{E}_d\,[\, L\,] = \frac{|R|}{q-p} - \frac{|R|+d}{q-p}\,\frac{1-(q/p)^{|R|}}{1-(q/p)^{|R|+d}}. \tag{8}$$

The expected cost of phase 2 can now be written as

$$T_2(d) = t_{\mathrm{b}}\, \mathrm{E}_d\,[\, L\,],$$

and the total resizing cost can be written as

$$T(d) = T_1(d) + T_2(d) = \mathrm{E}_d\,[\, t_{\mathrm{a}} g(U) + t_{\mathrm{b}} L\,].$$

*7.3.1 Finding an Optimum Parameterization*

We can now apply numerical methods to find the optimum value $d^*$ for $d$ so that $T(d)$ is minimized. The expected cost of phase 1 can be computed numerically, based on the formula

$$\mathrm{E}_d\,[\,g(U)\,] = \sum_{u=l}^{M'} g(u)\,\Pr\,[\,U = u\,] \tag{9}$$

where $\Pr\,[\,U = u\,]$ is given by (3) and $l = \max(M'-d, 0)$ denotes the minimum value of $U$ under our assumption that $M' < |R|$. The above sum can be evaluated quite efficiently because only a small number of terms contribute significantly to the sum. The expected cost of phase 2 can be computed using (8). Given both formulas, we can use standard numerical optimization algorithms to compute $d^*$. Under more complex cost models—e.g., when the goal is to minimize the probability that the resizing time exceeds a specified value or when a more sophisticated model of disk access is used—or under more complex stochastic models of the transaction stream, stochastic optimization techniques as described in [12, App. B] can be used to compute $d^*$.

We now explore a closed-form approximation to the function $T(d)$ that is highly accurate and agrees closely with our numerical results. This approximation immediately leads to an effective approximation of $d^*$. The first step in the approximation is to assume that $U = \mathrm{E}\,[\,U\,] = M'|R|/(|R|+d)$ with probability 1. Analogously to [12], our motivation is that the coefficient of variation

$$\mathrm{CV}[\,U\,] = \sqrt{\frac{\mathrm{Var}\,[\,U\,]}{\mathrm{E}^2\,[\,U\,]}} = \sqrt{\frac{d(|R|+d-M')}{M'|R|(|R|+d-1)}}$$

is of order $O(|R|^{-1/2})$, and $|R|$ is typically very large. Thus, $U$ will be close to its expected value with high probability.

Under the above assumption, the approximate expected phase 1 cost is

$$\begin{aligned}\hat{T}_1(d) &= g(\mathrm{E}\,[\,U\,]) = g\left(M'\frac{|R|}{|R|+d}\right)\\ &= t_\mathrm{a}|R|\,\ln\left[\frac{|R|-M}{|R|-M-N\left(M'\frac{|R|}{|R|+d}\right)}\right].\end{aligned}$$

Making use of the fact that $N(u) = 0$ for $u \le M$, and since

$$M'\frac{|R|}{|R|+d} \le M \quad \text{for} \quad d \ge |R|\frac{M'-M}{M},$$

we find that

$$\hat{T}_1(d) = \begin{cases} t_\mathrm{a}|R|\,\ln\left[\dfrac{(|R|-M)(|R|+d)}{|R|(|R|+d-M')}\right] & d < \theta\\ 0 & d \ge \theta,\end{cases}$$

Where $\theta = |R|(M'-M)/M$. The function $\hat{T}_1(d)$ is monotonically decreasing, convex, and differentiable on the interval $[0,\theta)$.

To approximate the expected phase 2 cost $T_2(d)$, we follow [12] and observe that the expected change of the dataset size after each transaction is $p\cdot 1+(1-p)\cdot(-1) = 2p-1$, so that the expected number of steps to increase the dataset size by 1 is roughly equal to $1/(2p-1)$. Thus, roughly $d/(2p-1)$ steps are required, on average, to increase the dataset size by $d$ and therefore to finish phase 2. This leads to an approximate expected phase 2 cost of

$$\hat{T}_2(d) = t_\mathrm{b}\frac{d}{2p-1}.$$

The above equation is precisely the limit of (8) as $|R| \to \infty$; in practice, the approximation is accurate when $|R|$ is not too small. The expected total time required to resize a sample is approximately equal to $\hat{T}(d) = \hat{T}_1(d) + \hat{T}_2(d)$.

We now choose $d = \hat{d}^*$, where $\hat{d}^*$ minimizes the function $\hat{T}$. Note that our search for $\hat{d}^*$ can be restricted to the interval $[0,\theta]$, because $\hat{T}_2(d)$ is increasing and $\hat{T}_1(d) = 0$ for $d \ge \theta$. Thus, to compute $\hat{d}^*$, first set

$$d_0 = \frac{M'}{2} - |R| + \sqrt{\frac{(M')^2}{4} + \frac{t_\mathrm{a}}{t_\mathrm{b}}|R|M'(2p-1)}. \tag{10}$$

If $d_0 \in [0,\theta)$, then $d_0$ satisfies $\hat{T}'(d_0) = 0$, and we take $\hat{d}^* = d_0$. Otherwise, we take $\hat{d}^*$ to be either 0 or $\theta$, depending upon which of the quantities $\hat{T}(0)$ or $\hat{T}(\theta)$ is smaller.

## 7.4 Example

We conducted a small experimental study to evaluate the performance of the RPR algorithm and provide additional insight into the effect of parameter $d$. The setup is the same as that for evaluating BR in [12]; we obtain very similar results, i.e., RPR is as efficient as BR but, in contrast, provides uniform samples.

Throughout, unless specified otherwise, we used initial and final sample sizes of $M$ = 100,000, $M'$ = 200,000, respectively, and also set $|R|$ = 1,000,000. In addition, we set $p = 0.6$ and $t_\mathrm{b}$ = 1ms; recall that $p$ represents the probability that a transaction is an insertion, and $t_\mathrm{b}$ is the expected time between arrivals

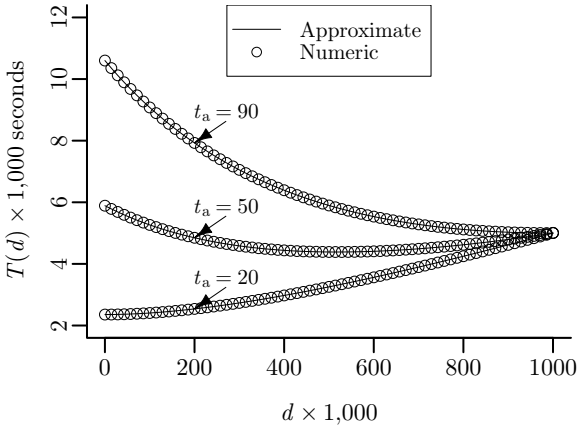


**Figure 2** Expected resizing cost $T(d)$

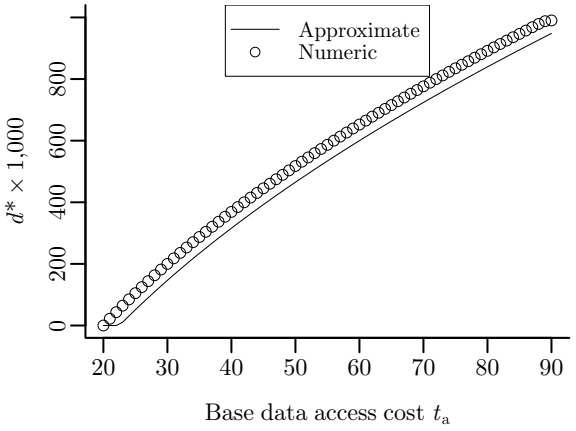


**Figure 3** Optimal value of $d$

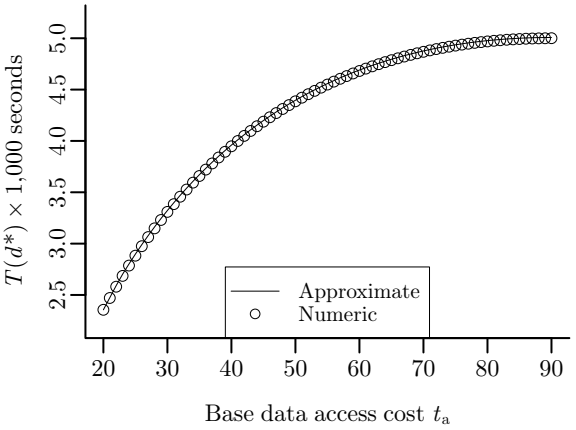


**Figure 4** Optimal expected resizing cost $T(d^*)$

during phase 2. The experimental results for various other choices of parameters were qualitatively similar.

Fig. 2 displays the expected resizing cost $T(d)$ for various values of $d$, when the base-data access cost $t_a$ equals 20, 50, and 90 milliseconds.[6] There are three possible behaviors of the $T$ function in the search interval: increasing, decreasing, and internal minimum point. Our choices of $t_a$ illustrate these three possible behaviors. For each scenario, the approximate cost $\hat{T}$ is represented as a solid curve. Superimposed on this curve are points that represent the exact (expected) cost $T$ for various values of $d$. As expected, when the base-data access cost $t_a$ is relatively small, the cost function achieves its minimum value at $d = 0$, and the optimal strategy is to increase the sample size to $M'$ during phase 1, and not execute phase 2. When $t_a$ is relatively large, the cost function achieves its minimum value at $d = 1{,}000{,}000$, and the optimal strategy is to not sample the base data at all, and increase the sample size to $M'$ exclusively during phase 2. For an intermediate value of $t_a$, the optimal value of $d$ falls in between 0 and 1,000,000—here, $d^* \approx 517{,}745$ for $t_a = 50$ms—so that the resizing work is allocated between the two phases. Note that, in this example, the expected costs corresponding to the best and worst choices of $d$ can vary by a factor of two. Moreover, the approximate and exact costs are extremely close to each other. This high degree of consistency, which was observed for all parameter values that we investigated, increases our confidence in the quick approximate cost model.

The high accuracy of the cost approximation leads us to expect that our numerical and approximate methods will also yield similar estimates for $d^*$. This expectation is fulfilled, as shown in Figs. 3 and 4. Fig. 3 shows the optimal value $d^*$ for various values of $t_a$, while Fig. 4 shows the (exact) expected resizing cost for the numeric and approximate value of $d^*$. As before, the solid line represents values computed via the quick approximate method and the circular points represent the numerical solutions. The approximate method seems to slightly underestimate the exact value of $d^*$. However, even when the value of $d^*$ produced by the numerical method differs slightly from the result of the approximate closed-form model, the resulting resizing costs do not differ perceptibly. The reason is that the cost curve—as shown in Fig. 2—is flat around the optimum value of $d^*$.

To evaluate the stability of RPR with respect to its performance, we run as a final experiment 100 independent repetitions of a Java implementation of RPR. We used the three scenarios $t_a = 20$, $t_a = 50$ and $t_a = 90$ and set $d$ to its respective optimum value. Within each run, we monitored the number of base data accesses and the total number of arriving transactions until the resizing process ended. The results are given in Fig. 5,

[6] A complete recomputation of the sample from scratch therefore takes 4,000$s$, 10,000$s$ and 18,000$s$, respectively.

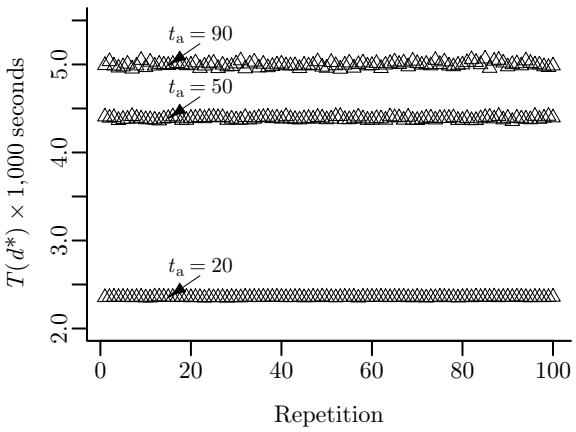


**Figure 5** Actual resizing cost at optimum parameterization

where we print a symbol for each individual run. As can be seen, the running times are very stable, with little variation across multiple runs. We conclude that the actual running time of the RPR algorithm stays close to its expected value with high probability, so that RPR exhibits a stable performance.

## 8 Practical considerations

In this section, we provide a simple technique for testing an implementation of a sampling scheme for uniformity. We also discuss the impact of non-uniformity in practice.

### 8.1 Testing "uniform" sampling schemes

We propose a simple statistical test for detecting whether a given sampling scheme $\mathscr{A}$ is non-uniform. In brief, we first run $\mathscr{A}$ many times on some small transaction sequence $\gamma$, and then run a chi-squared test on the expected and observed frequencies of all possible samples. If $\mathscr{A}$ fails the test, it is very likely to be non-uniform. If $\mathscr{A}$ passes, no conclusion can be drawn. We found, however, that most of the non-uniform schemes discussed in this article do not pass our statistical test.

In more detail, denote by $\gamma$ denote a (small) sequence of insertions and transactions and denote by $R$ the corresponding dataset. We now perform many independent runs of $\mathscr{A}$ on $\gamma$ to obtain a multiset $\mathscr{S} = \{ S_1, \ldots, S_l \}$ of samples. For every sample size $n$ in $\{ 0, \ldots, |R| \}$, denote by $\mathscr{S}_n = \{ S_{n,1}, \ldots, S_{n,l_n} \}$ the multiset of samples in $\mathscr{S}$ of size exactly $n$. Finally, denote by $\mathscr{R}_n$ the set of all size-$n$ subsets of $R$ (using multiset semantics). For each $A \in \mathscr{R}_n$, denote by $p_n(A) = l_n \Pr[\, S = A \mid |S| = n \,]$ the expected frequency of $A$; from Eq. (1), we obtain

$$p_n(A) = l_n \binom{|R|}{n}^{-1} \prod_{r \in D(A)} \binom{|R(r)|}{|A(r)|}.$$

Finally, denote by $f_n(A)$ the observed frequency of $A$ in $\mathscr{S}_n$.

Our test is as follows: If $D(\mathscr{S}_n) \setminus \mathscr{R}_n \neq \emptyset$, then $\mathscr{A}$ is clearly non-uniform: it produces a size-$n$ "sample" that is not a size-$n$ subset of $\mathscr{R}$. Otherwise, set $\boldsymbol{p}_n = (p_n(A) : A \in \mathscr{R}_n)$ and $\boldsymbol{f}_n = (f_n(A) : A \in \mathscr{R}_n)$. We now run a chi-squared test with the null hypothesis that $\mathscr{A}$ is uniform for size-$n$ samples, i.e., $\boldsymbol{f}_n$ is consistent with $\boldsymbol{p}_n$. If the test fails for some $n$, then $\mathscr{A}$ is likely to be non-uniform. For convenience, a Python implementation of our test is provided at `http://www.mpi-inf.mpg.de/~rgemulla/code/uniformity-testing.tgz`.

### 8.2 Impact of non-uniformity

To shed some light on the degree of non-uniformity in practice, we investigated the non-uniformity introduced by HBS$(q, M)$. In particular, we created a sequence of insertions $\gamma_N = (+r_1, +r_2, \ldots, +r_N)$ and computed the exact marginal inclusion probability $\Pr[\, r \in S_N \,]$ of each $r \in \{ r_1, \ldots, r_N \}$ in an HBS$(q, M)$ sample. If HBS$(q, M)$ were truly uniform, the marginal inclusion probabilities would be identical for all items (the opposite does not hold).

For $1 \leq N \leq 50$, Fig. 6 plots the inclusion probabilities of each item as obtained by HBS$(0.5, 10)$ conditioned on a switch-over point of $T(M) = 20$, i.e., conditioned on the event that we switch to reservoir sampling after exactly 20 insertions. The $x$-axis in the figure corresponds to the number $N$ of insertions. Each line corresponds to one item and is printed transparently (darker regions thus indicate that many items have the same marginal inclusion probability); the line starts when the item is inserted. First, observe that after 20 insertions, item $r_{20}$ is present in the sample with probability 1 (the top-most line); this is because we conditioned on the event $T(M) = 20$ and $r_{T(M)} \in S_{T(M)}$. After all insertions have been processed, $r_{20}$ has the highest inclusion probability (overrepresented), items $r_1, \ldots, r_{19}$ have lowest inclusion probability (underrepresented), and items $r_{21}, \ldots, r_{50}$ have intermediate inclusion probability (correctly represented).

Fig. 7 shows the unconditional marginal inclusion probabilities of HBS$(0.5, 10)$. As can be seen, there is still a wide spread of inclusion probabilities. Again, early items are underrepresented, items around the expected switch-over time are overrepresented, and the

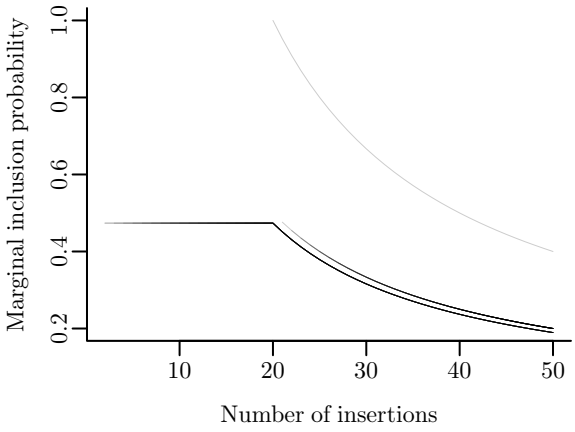


**Figure 6** Marginal inclusion probabilities of HBS(0.5, 10) sampling conditioned on switch-over point $T(M) = 20$ (50 insertions)

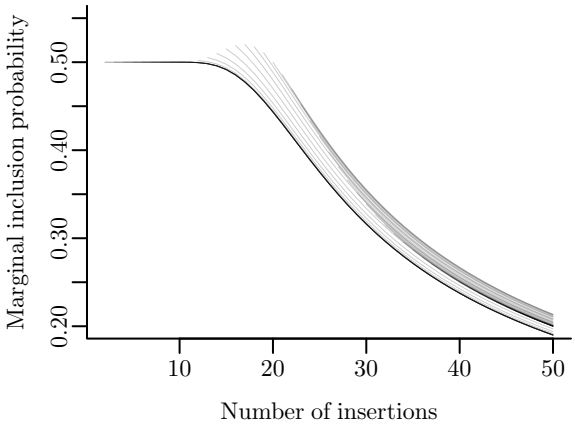


**Figure 7** Marginal inclusion probabilities of HBS(0.5, 10) sampling (50 insertions)

marginal inclusion probabilities of late items converge to the correct value (as $N \to \infty$). In Fig. 8, we show a similar plot for a larger sample size, i.e., HBS(0.5, 100) (only every 10th item is shown). As can be seen, the spread of inclusion probabilities decreased significantly: When the sample size is large, the non-uniformity of the HBS switch-over is spread over more items so that the impact per item is smaller. Thus thus the larger the sample size, the lower the effect of non-uniformity.

Even though the degree of non-uniformity in terms of marginal inclusion probabilities appears to vanish as the sample size grows, the workaround HBSR$(q, M)$ has virtually the same costs as HBS$(q, M)$ but always guarantees uniformity. Thus HBSR$(q, M)$ is manifestly preferable to HBS$(q, M)$. Similar arguments hold for most of the workarounds proposed in this article.

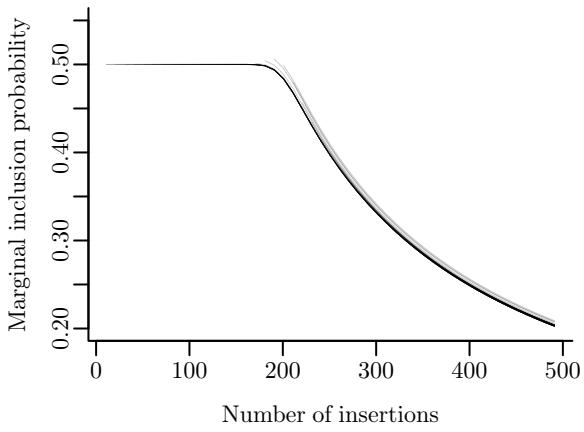


**Figure 8** Marginal inclusion probabilities of HBS(0.5, 100) sample (500 insertions)

## 9 Summary and Guidelines

We have shown that a number of previously proposed "uniform" bounded sampling schemes actually fail to be uniform, often in subtle ways. Errors creep in when switching from a Bernoulli-sampling scheme to a reservoir-sampling scheme, and when treating a non-Bernoulli sample as if it were Bernoulli. Attempts to directly bound the sample footprint can also lead to problems, as can the use of hash functions that are not fully independent. Although the effects of non-uniformity are often small, they are unpredictable. Moreover, they can be easily detected by using simple statistical tests, and can be easily avoided by using a variety of workarounds such as randomized switching, relaxation to tightly controlled probabilistic bounds, and use of independent hash functions.

A key question facing the practitioner is which sampling methods to use in different scenarios. Our results lead to the following guidelines.

*Set sampling:* RP suffices for most applications, since it enforces a strict sample-size bound, and supports both insertion-only and general UDI transaction sequences. Moreover, samples can be merged using RP-MERGE to facilitate distributed sampling, and can be resized using the RPR algorithm. In insertion-only environments, the BSP$(q_0, M)$ algorithm is dominated by the HBSR$(q, M)$ algorithm, which maintains more stable sample sizes. Given the superior properties of RP, however, both of these algorithms are of more theoretical than practical interest. The PBS$(M, \delta)$ algorithm may be useful in distributed sampling settings, since it maintains Bernoulli samples that can be easily merged (and incrementally maintained, if desired). The downside of this latter algorithm is that only probabilistic bounds, and not strict bounds, are enforced.

*Multiset sampling:* In insertion-only settings, the RP algorithm (which reduces to reservoir sampling) is superior for the reasons outlined above. In the presence of deletions, the PABS$(F, \delta)$ algorithm is the only known option, and hence only probabilistic sample-size bounds seem possible.

*Distinct-item sampling:* In insertion-only settings, the MIN$(M)$ algorithm is superior to the UDI algorithms since it provides a stable sample size, a stable footprint, and has low computational costs. In the UDI setting, AKMV$(F)$ appears to be the method of choice; it provides a strict sample size and footprint bound and is almost as computationally efficient as MIN$(M)$. Since AKMV$(F)$ samples can also be merged efficiently, the scheme is superior to DPBS$(F, \delta)$ (only probabilistic bounds) and DIS$_\infty(F)$ (large memory consumption and runtime costs). All of the above methods make use of an $F$-wise independent or fully independent hash function, which may be too expensive to use in practice. An alternative might be to use cryptographic hash functions instead; experiments in [10] suggest that the samples obtained in this manner still pass statistical tests for uniformity, but correctness is not guaranteed. These problems are avoided by PABSD$(F, \delta)$, which does not make use of hashing; disadvantages of PABSD$(F, \delta)$ are, however, that it provides only probabilistic bounds on the sample footprint, and that the obtained distinct-item sample is generally smaller than that of AKMV$(F)$ and DPBS$(F)$. When a multiset sample needs to be maintained in addition to the distinct-item sample, PABSD$(F, \delta)$ is attractive: The PABSD$(F, \delta)$ sample can be obtained for free from a PABS$(F, \delta)$ multiset sample.

## References


1. Kevin Beyer, Peter J. Haas, Berthold Reinwald, Yannis Sismanis, and Rainer Gemulla. On synopses for distinct-value estimation under multiset operations. In *Proc. of the 2007 ACM SIGMOD Intl. Conf. on Management of Data*, pages 199–210, 2007.
2. A. Broder. On the resemblance and containment of documents. In *Proc. of the Compression and Complexity of Sequences 1997*, pages 21–29, 1997.
3. Andrei Z. Broder, M. Charikar, A.M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.
4. Paul G. Brown and Peter J. Haas. Techniques for warehousing of sample data. In *Proc. of the 2006 Intl. Conf. on Data Engineering*, page 6, 2006.
5. Edith Cohen, Graham Cormode, and Nick Duffield. Don't let the negatives bring you down: Sampling form strams of signed updates. In *Proc. ACM SIGMETRICS*, pages 343–354, 2012.
6. Graham Cormode, S. Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proc. of the 2005 Intl. Conf. on Very Large Data Bases*, pages 25–36, 2005.
7. C. T. Fan, M. E. Muller, and I. Rezucha. Development of sampling plans by using sequential (item by item) selection techniques and digital computers. *J. Amer. Statist. Assoc.*, 57:387–402, 1962.
8. William Feller. *An Introduction to Probability Theory and Its Applications*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 3rd edition, 1968.
9. Alan M. Ferrenberg and D. P. Landau. Monte Carlo simulations: Hidden errors from "good" random number generators. *Phys. Rev. Lett.*, 69(23):3382–3384, 1992.
10. Rainer Gemulla. *Sampling Algorithms for Evolving Datasets*. PhD thesis, Technische Universität Dresden, 2008. `http://nbn-resolving.de/urn:nbn:de:bsz:14-ds-1224861856184-11644`.
11. Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. Maintaining Bernoulli samples over evolving multisets. In *Proc. of the 2007 ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*, pages 93–102, 2007.
12. Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. Maintaining bounded-size sample synopses of evolving datasets. *The VLDB Journal*, 17(2):173–201, 2008.
13. Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*, pages 331–342, 1998.
14. A. I. McLeod and D. R. Bellhouse. A convenient algorithm for drawing a simple random sample. *Appl. Statist.*, 32:182–184, 1983.
15. Yufei Tao, Xiang Lian, Dimitris Papadias, and Marios Hadjieleftheriou. Random sampling for continuous streams with arbitrary updates. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):96–110, 2007.
16. M. N Wegman and J. L Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.


## Appendix: Proof of Theorem 1

First note that, in the first two assertions of the theorem, $A \subseteq R_k$ implies that $|A| \leq k$, so that expressions such as $k - |A|$ are always non-negative. Next observe that $T = l$ if and only if exactly $M$ items are accepted into the sample during the first $l - 1$ steps of Bernoulli sampling and then an item is accepted at the $l$th step. The probability of the first event is binomial: $\binom{l-1}{M} q^M (1-q)^{l-1-M}$. The probability of the second event is $q$. Because successive steps of Bernoulli sampling are independent, the joint probability of the two events is simply the product of the individual probabilities, which yields the definition of $\pi_k$ given in the theorem.

To prove the first assertion of the theorem, fix $k \geq M + 1$ and $A \subseteq R_k$ with $|A| = M$, and observe that

$$\begin{aligned}\Pr[S_k = A, T \leq k] &= \sum_{l=M+1}^{k} \Pr[S_k = A, T = l] \\ &= \sum_{l=M+1}^{k} \Pr[S_k = A \mid T = l]\, \pi_l.\end{aligned}$$

Straightforward generalization of the reasoning given in the example after the statement of the theorem shows that $\Pr[S_k = A \mid T = l] = \binom{k}{|A|}^{-1} = \binom{k}{M}^{-1}$ for each value of $l$, so that

$$\begin{aligned}\Pr[S_k = A, T \le k] &= \binom{k}{M}^{-1} \sum_{l=M+1}^{k} \pi_l \\ &= \binom{k}{M}^{-1} \Pr[T \le k].\end{aligned}$$

Dividing through by $\Pr[T \le k]$ yields the desired result.

To prove the second assertion, first assume that $|A| < M$. Denote by $B_1, B_2, \ldots,$ the sequence of samples obtained by applying pure Bernoulli sampling to the transaction stream. Observe that the event $\{S_k = A\}$ implies the event $\{T > k\}$, so that $\Pr[S_k = A, T > k] = \Pr[B_k = A]$. It follows that

$$\Pr[S_k = A, T > k] = q^{|A|}(1-q)^{k-|A|}. \tag{11}$$

Now assume that $|A| = M$. It is no longer true that $S_k = A$ implies $T > k$, but it is still true that the event $\{ S_k = A, T > k \}$ occurs if and only if RS is still in Bernoulli sampling mode at the $k$th step and $S_k = A$, so that (11) holds for this case also. Dividing both sides of (11) by $\Pr[T > k] = \sum_{l>k} \pi_l$ yields the desired result.

Combining these results, we have, for any $k \ge 1$ and $A \subseteq R_k$,

$$\begin{aligned}&\Pr[S_k = A] = \Pr[S_k = A, T \le k] + \Pr[S_k = A, T > k] \\ &= \begin{cases} q^{|A|}(1-q)^{k-|A|} & \text{if } |A| < M \\ \binom{k}{|A|}^{-1} \sum_{l=M+1}^{k} \pi_l + q^{|A|}(1-q)^{k-|A|} & \text{if } |A| = M. \end{cases}\end{aligned}$$

Because the right side depends on $A$ only through $|A|$, the third assertion of the theorem follows.